独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：刘恩龙　　日期：2010.6.6

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：刘恩龙　　日期：2010.6.6

导师签名：[illegible]　　日期：2010.6.10

指纹 FPGA 粗比对加速卡的优化

摘 要

本文首先对指纹粗比对加速卡的 FPGA 核心比对模块做了改进，消除了原设计在特殊情况下出现的一些问题，并优化了原设计的时序。对修改后的核心比对模块进行后仿真后，将新的代码下载到 Altera 公司的 StratixII FPGA 内，用 QuartusII 的嵌入式逻辑分析仪观察，确认能正常工作。

核心电路完成粗比对后，将与现场指纹匹配的库指纹序号返回主机。本文提出了一种利用原设计中库指纹空闲空间存放库指纹序号的方法，具有不改变原设计时序、开发难度低的优点。为提高系统性能，匹配结果被暂存在一个队列里，等到队列充满到一定程度再发中断通知 9054 从队列中读出比对结果。在调试阶段，队列用片内 RAM 实现，如有必要也可采用专用芯片。

本课题的 FPGA 的片内存储资源不够存放从 PCI 9054 一次发送的库指纹数据，因此外接了 4 片 IDT 公司的 IDT71V SRAM 芯片作为外部存储器，本文设计了对 SRAM 的读写控制。

PLX 公司的 9054 芯片是兼容 PCI 2.2 版本的总线桥设备，在本课题里，用来将复杂的 PCI 协议转化为相对简单的本地总线协议，降低开发难度。本文设计了与 9054 的接口。

本文的最后一项工作是进行主机端应用程序的设计和应用程序、驱动和加速卡的联合调试，并在结尾给出总结与展望。

关键字：粗比对　FPGA　SRAM　9054

The Optimization of the Fingerprint FPGA Coarse Matching Accelerator

ABSTRACT

This paper first makes some improvements to the FPGA core of fingerprint coarse matching card, eliminating some problems in the original design which may occur in some odd circumstances, and optimazing the timing of the original design. The paper did some timing-simulation to the revised core matching module first, then downloaded the new code to Altera's Stratix II FPGA, and observed the waveform using Quartus II's embedded logic analyzer, thus verified the new code work well.

After the coarse matching, the core returns the ID of library fingerprints, which accord with the field fingerprint, to the computer. This paper puts forward a method which utiliźes the former free space of the library fingerprint in the original design. This method has such advantages as keep the original timing and easy to deveop. In order to raise the system's performance, the matching result is temporily stored in a queue. When the queue is stuffed to a certain extant, an interrupt is sent to inform 9054 to read the matching result from the queue. In the debugging phase, the queue is fulfilled with on-chip RAM, however, it can also be an dedicated chip if necessary.

In this issue, the FPGA has not enough memory to store one batch of library fingerprints sent from PCI 9054, so it's connected to 4 IDT's IDT71V SRAM chips as external memory. This paper makes an presentation about the reading and writing of SRAM.

The PLX's 9054 is a bus bridge compatible with PCI version 2.2. In this issue, it is used to turn complicated PCI protecol into simpler local bus protecol. This way the design is easier. This paper makes an presentation about the design of the interface to 9054.

The last part of the paper is about the design of the application in the computer and the joint test of the application, the driver and the accelerator. A summary and prospect of the issue is given at the end of the paper.

KEYWORDS: Coarse matching, FPGA, 9054, SRAM

目录

第一章 绪论

1.1 指纹粗比对加速卡的研究背景

指纹因其唯一性和终生不变性成为目前最可靠的生物识别技术的研究对象。指纹的终生不变性指的是人的指纹纹线结构和细节特征终生不会发生变化；指纹的唯一性是说世界上没有任何两个人的指纹相同。

指纹是人的手指末端正面皮肤上凸凹不平产生的纹线。纹线有规律的排列形成不同的纹型。纹线的起点、终点、结合点和分叉点，称为指纹的细节特征点（minutiae）。指纹识别即指通过比较不同指纹的细节特征点来进行身份鉴别的技术。

指纹的应用历史悠久，经考古证实，公元前 7000 年到 6000 年以前，在古叙利亚和中国，指纹作为身份鉴别的一种手段，已经得到应用。但长期以来，人们对指纹的应用还停留在原始阶段，主要作为身份凭证，例如，我国古代利用指纹（手印）来签字画押。直到 1684 年，英国植物形态学家 Grew 发表了第一篇研究指纹的科学论文，标志着现代意义的指纹识别理论的诞生。之后指纹识别理论蓬勃发展。

1809 年 Bewick 把自己的指纹作为商标。1823 年解剖学家 Purkije 将指纹分为九类。 1880 年，Faulds 在《自然》杂志提倡将指纹用于识别罪犯。1891 年 Galton 提出著名的高尔顿分类系统。之后，英国、美国、德国等的警察部门先后采用指纹鉴别法作为身份鉴定的主要方法。随着计算机和信息技术的发展，FBI 和法国巴黎警察局于六十年代开始研究开发指纹自动识别系统（AFIS）用于刑事案件侦破。目前，世界各地的警察局已经广泛采用了指纹自动识别系统。九十年代，用于个人身份鉴定的自动指纹识别系统得到开发和应用。以上事件在指纹识别的发展史上都具有里程碑的意义。

飞速发展的信息技术将人类社会带入一个崭新的信息时代。在高度信息化的今天，信息安全变得尤为重要，人们对身份验证和识别的需求也越发迫切。生物特征识别技术很好地满足了这一需求。可用于识别的生物特征有指纹、掌纹、人脸、视网膜、虹膜、毛发等。其中指纹识别技术是最传统、发展最成熟也是应用最广泛的生物特征识别技术。

和其他生物特征识别方式比较，指纹识别有如下特点：

表 1-1[1]

生物统计特征	普遍性	唯一性	稳定性	可采集性	准确性	可接受性	安全性
指纹	中	高	高	中	高	中	高
脸型	高	低	中	高	低	高	低
手形	中	中	中	高	中	中	中
击键方式	低	低	低	中	低	中	中
虹膜	高	高	高	中	高	低	高
DNA	高	高	高	低	高	低	低

从表中可见，指纹的各项指标都较好。和其他生物特征识别方式比较，指纹有内在的高唯一性——每个人的指纹都是独一无二的；高稳定性——指纹的形状特征固定不变。；高安全性——每个人有多枚指纹可用；高准确性——指纹识别技术理论严谨、技术成熟；其他特性也较好，因此是首选的生物特征识别手段。

根据指纹应用场合的不同，指纹识别系统可分为身份认证和查证两大类。身份认证系统是一对一的查比系统，即将欲验证的用户指纹与库中存储的合法用户的指纹进行比对来确认其身份的合法性；而查证系统则是一对多的查比系统，警察局和公安部门就是应用查证系统来进行案件侦破的。这两种系统有截然不同的特点：前者只需验证欲比对指纹和特定库指纹是否匹配；后者则需要搜索和遍历数据库中的每一枚指纹，将欲比对的指纹和库中指纹进行一一比对[2]。

在国际上，自动指纹识别系统（AFIS）的研究开始于二十世纪六十年代。在近五十年的发展中，先后有多种规模、性能不同的系统问世。

指纹比对是重复的大运算量任务，随着近年来应用的深入与推广，指纹比对的规模更是不断增加。用硬件设备完成指纹比对，集中主机处理能力于其他的工作是一个比较好的对策。配备专用的指纹比对加速卡不仅可以大幅度提高系统的速度，同时也为系统中采用更复杂的算法来提高系统处理的精度创造了更有利的条件。由于指纹库中的指纹数目巨大，把现场指纹和它们一一进行比对，要耗费大量的时间，因此在精细的比对之前，先进行一次粗比对，排除大量的明显不匹配的指纹是提高自动指纹识别系统的一个有效途径。

1.2 主要工作和章节安排

本论文针对指纹粗比对加速卡，主要完成了以下 4 个工作：

（1） 优化和改进了粗比对算法在 FPGA 上的实现，修正了原设计的一些问题，使得算法在 FPGA 上的工作更加稳定可靠；并进行了功能仿真验证。

（2） 设计了外部 SRAM 控制器，通过用嵌入式逻辑分析仪调试，验证了该控制器可以正常工作。

（3） 设计了 FPGA 与 9054 芯片的接口，修改了 PCB 布线，并用嵌入式逻辑分析仪调试，验证了该电路工作正常。

（4） 修改了驱动程序和应用程序，使应用程序、驱动程序和加速卡可以协同工作，并经实验验证了其工作正常。

本论文共分 7 章，第 1 章介绍了自动指纹识别系统的研究背景，以及论文工作和章节安排；第 2 章介绍了 VHDL 语言、FPGA 开发基本原则、技巧及 StratixII 系列芯片的内部结构特性；第 3 章是对原粗比对算法的优化和改进；第 4 章介绍 SRAM 控制器的设计；第 5 章阐述了与 9054 芯片的接口的设计；第 6 章给出了应用程序、驱动程序和加速卡联合工作的测试结果；第 7 章对指纹粗比对系统进行总结和展望。

第二章 硬件描述语言 VHDL 与 FPGA 设计

2.1 VHDL 语言的特性及优势

VHDL 语言是一种用于电路设计的高级语言。它在 20 世纪 80 年代后期出现。最初是由美国国防部开发出来供美军用来提高设计的可靠性和缩减开发周期的一种使用范围较小的设计语言。1987 年底，VHDL 被 IEEE 和美国国防部确认为标准硬件描述语言。自 IEEE 公布了 VHDL 的标准版本，IEEE-1076（简称 87 版)之后，各 EDA 公司相继推出了自己的 VHDL 设计环境，或宣布自己的设计工具可以和 VHDL 接口。此后 VHDL 在电子设计领域得到了广泛的接受，并逐步取代了原有的非标准的硬件描述语言。1993 年，IEEE 对 VHDL 进行了修订，从更高的抽象层次和系统描述能力上扩展 VHDL 的内容，公布了新版本的 VHDL，即 IEEE 标准的 1076-1993 版本，简称 93 版。2008 年，在 VHPI 任务团队和 P1076 工作组的共同努力下，推出 VHDL 2008。现在，VHDL 和 Verilog 作为 IEEE 的工业标准硬件描述语言，又得到众多 EDA 公司的支持，在电子工程领域，已成为事实上的通用硬件描述语言。

VHDL 的英文全称是 VHSIC（Very High Speed Integrated CircuitHardware Description Language），翻译成中文就是超高速集成电路硬件描述语言。因此它的应用主要是应用在数字电路的设计中。目前，它在中国的应用多数是用在 FPGA/CPLD/EPLD 的设计中。当然在一些实力较为雄厚的单位，它也被用来设计 ASIC。

VHDL 主要用于描述数字系统的结构、行为、功能和接口。除了含有许多具有硬件特征的语句外，VHDL 的语言形式、描述风格与句法十分类似于一般的计算机高级语言。VHDL 的程序结构特点是将一项工程设计，或称设计实体（可以是一个元件、一个电路模块或一个系统）分成外部（或称可视部分及端口)和内部（或称不可视部分），既涉及实体的内部功能和算法完成部分。在对一个设计实体定义了外部界面后，一旦其内部开发完成，其他的设计就可以直接调用这个实体。这种将设计实体分成内外部分的概念是 VHDL 系统设计的基本点。

VHDL 语言具有很强的电路描述和建模能力，能从多个层次对数字系统进行建模和描述，从而大大简化了硬件设计任务，提高了设计效率和可靠性。

VHDL 具有与具体硬件电路无关和与设计平台无关的特性，具有良好的电路行为描述和系统描述的能力，并在语言易读性和层次化、结构化设计方面，表现了强大的生命力和应用潜力[3]。

与其他硬件描述语言相比，VHDL 具有以下特点：

功能强大、设计灵活。VHDL 具有功能强大的语言结构，可以用简洁明确的源代码来描述复杂的逻辑控制。它具有多层次的设计描述功能，层层细化，最后可直接生成电路级描述。VHDL 支持同步电路、异步电路和随机电路的设计，这是其他硬件描述语言所不能比拟的。VHDL 还支持各种设计方法，既支持自底向上的设计，又支持自顶向下的设计；既支持模块化设计，又支持层次化设计。

支持广泛、易于修改。由于 VHDL 已经成为 IEEE 标准所规范的硬件描述语言，目前大多数 EDA 工具几乎都支持 VHDL，这为 VHDL 的进一步推广和广泛应用奠定了基础。在硬件电路设计过程中，主要的设计文件是用 VHDL 编写的源代码，因为 VHDL 易读和结构化，所以易于修改设计。

强大的系统硬件描述能力。VHDL 具有多层次的设计描述功能，既可以描述系统级电路，又可以描述门级电路。而描述既可以采用行为描述、寄存器传输描述或结构描述，也可以采用三者混合的混合级描述。另外，VHDL 支持惯性延迟和传输延迟，还可以准确地建立硬件电路模型。VHDL 支持预定义的和自定义的数据类型，给硬件描述带来较大的自由度，使设计人员能够方便地创建高层次的系统模型。

独立于器件的设计、与工艺无关。设计人员用 VHDL 进行设计时，不需要首先考虑选择完成设计的器件，就可以集中精力进行设计的优化。当设计描述完成后，可以用多种不同的器件结构来实现其功能。

很强的移植能力。VHDL 是一种标准化的硬件描述语言，同一个设计描述可以被不同的工具所支持，使得设计描述的移植成为可能。

易于共享和复用。VHDL 采用基于库（Library）的设计方法，可以建立各种可再次利用的模块。这些模块可以预先设计或使用以前设计中的存档模块，将这些模块存放到库中，就可以在以后的设计中进行复用，可以使设计成果在设计人员之间进行交流和共享，减少硬件电路设计的时间。

FPGA 设计不同于软件设计，它有着自己特有的设计原则，下面是 FPGA 常用的设计原则，在进行 FPGA 设计时，必须遵照这些基本原则。

2.3 开发 FPGA 的原则和技巧

2.3.1 开发 FPGA 的几点原则

1. 面积和速度的平衡与互换原则

这里"面积"指一个设计消耗 FPGA/CPLD 的逻辑资源的数量，对于 FPGA 可以用所消耗的触发器和查找表来衡量，更一般的衡量方式可以使用设计所占用的等价逻辑门数；"速度"指设计在芯片上稳定运行时所能够达到的最高频率，这个频率由设计的时序状况决定，和设计满足的时钟周期、管脚到管脚延时（PAD to PAD Time）、建立时间（Clock Setup Time）、保持时间（Clock Hold Time）以及时钟到输出延时（Clock to Output Delay）等众多时序特征量密切相关。面积与速度这两个指标贯穿着 FPGA/CPLD 设计的始终，是设计质量评价的终极标准。

面积和速度是一对对立统一的矛盾体。要求一个同时具备设计面积最小，运行频率最高是不现实的。科学的设计目标应该是在满足设计时序要求（包含对设计频率的要求）的前提下，占用最小的芯片面积。或者在所规定的面积下，使设计的时序余量更大，频率更高。这两种目标充分体现了面积和速度的平衡思想。关于面积和速度的要求，我们不应该简单地理解为工程师水平的提高和设计完美性的追求，而应该认识到它们是和产品的质量和成本直接相关的。如果设计的时序余量比较大，运行的频率比较高，则意味着设计的健壮性更强，整个系统的质量更有保证；另一方面，设计所消耗的面积更小，则意味着在单位芯片上实现的功能模块更多，需要的芯片数量更少，整个系统的成本也随之大幅度削减[4]。

面积和速度的互换是 FPGA/CPLD 设计的一个重要思想。从理论上讲，一个设计如果时序余量较大，所能跑的频率远远高于设计要求，那么就能通过功能模块复用减少整个设计消耗的芯片面积，这就是用速度的优势换面积的节约；反之，如果一个设计的时序要求很高，普通方法达不到设计频率，那么一般可以通过将数据流串并转换，并行复制多个操作模块，对整个设计采取"乒乓操作"和"串并转换"的思想进行运作，在芯片输出模块再对数据进行"并串转换"，使得从宏观上看整个芯片满足了处理速度的要求，这相当于用面积复制换取速度的提高。

2. 硬件原则

硬件原则主要针对 HDL 代码编写而言。

首先应该明确 FPGA/CPLD、ASIC 的逻辑设计所采用的硬件描述语言（HDL）与软件语言（如 C，C++等）是有本质区别的，以 VeilogHDL 语言为例，虽然 Veilog 很多语法规则和 C 语言相似，但是 Veilog 作为硬件描述语言，它的本质作用在于描述硬件，而不是设计硬件，它的最终实现结果是芯片内部的实际电路。所以评判一段 HDL 代码的优劣的最终标准是：其描述并实现的电路的性能。片面追求代码的整洁、简短是错误的，是与评价 HDL 的标准背道而驰的，正确的编码方法是，首先要做到对所需实现的硬件电路非常熟悉，对该部分硬件的结构与连接十分清晰，然后再用适当的 HDL 语句表达出来才可以。

3. 系统原则

系统原则包含两个层次的含义：具体到FPGA设计层面，要求对设计的全局有个宏观上的合理安排，比如时钟域、模块复用、约束、面积、速度等问题，要知道在系统上复用模块节省的面积远比在代码上小打小闹来的实惠得多；从更高层面上看，是一个硬件系统，一块单板如何进行模块花费与任务分配，什么样的算法和功能适合放在FPGA里面实现，什么样的算法和功能适合放在DSP、CPU里面实现，以及FPGA的规模估算、数据接口设计等。

一般来说，实时性要求高、频率快的功能模块适合使用FPGA/CPLD实现。与CPLD相比，FPGA更适合实现规模较大、频率较高、寄存器资源使用较多的设计。使用FPGA/CPLD设计时，应该对芯片内部的各种底层硬件资源，和可用的设计资源有一个较深刻的认识。如FPGA通常触发器资源比较丰富，而CPLD组合逻辑资源更多一些。

4. 同步设计原则

采用同步时序设计是FPGA/CPLD设计的一个重要原则，在FPGA器件中，有非常丰富的触发器资源和全局时钟资源，可以很方便地进行同步设计，利用触发器构成同步电路，采用全局时钟同步，将会给设计带来极大方便，使大规模设计成为可能。

采用同步设计方法可以有效避免竞争和冒险，提高电路运行的稳定性，极大减少亚稳态出现的几率；同步设计中，采用流水线的方法，对逻辑进行流水线分级，各级之间互不干扰，提高了硬件资源的利用效率，同时也减少了系统的时延；在高速电路中，采用同步设计的方法可以减少电平翻转的频率，有效降低系统的功耗。

2.3.2 开发FPGA的一些技巧

1. 流水线设计

流水线处理是高速设计中的一个常用设计手段。如果某个设计的处理流程分为若干步骤，而且整个数据处理是“单流向”的，即没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入，则可以考虑采用流水线设计方法提高系统的工作频率，即将比较大的组合逻辑块分成小块实现，小逻辑块之间用触发器间隔，如图2-1所示为流水线划分的示意图。

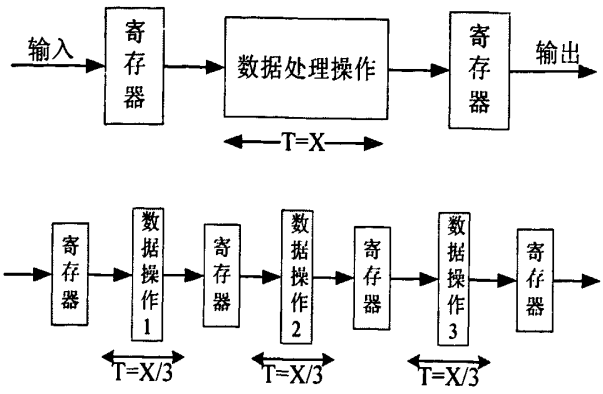

图 2-1 流水线划分示意图[5]

流水线设计的一个关键在于：整个设计时序的合理安排、前后级接口间数据流速的匹配。这就要求每个操作步骤的划分必须合理，要统筹考虑各个操作步骤间的数据流量。如果前级操作时间恰好等于后级的操作时间，设计最为简单，前级的输出直接汇入后级的输入即可。如果前级操作时间小于后级的操作时间，则需要对前级的输出数据适当缓存，才能汇入后级，还必须注意数据速率的匹配，防止后级数据的溢出。如果前级操作时间大于后级的操作时间，则必须通过逻辑复制、串并转换等手段将数据流分流，或者在前级对数据采用存储、后处理方式，否则会造成与后级的处理节拍不匹配。

事实上，在设计中加入了流水线，尽管可以提高系统的工作频率，但并不会减少原设计中总的延时，有时甚至还会增加插入的寄存器的延时及信号同步的时间差， 而且流水线的级数过多，虽然系统的工作频率可能会很高，但是系统的工作效率不一定高，所以，采用流水线设计方法要权衡利弊。

2. 乒乓操作

乒乓存储操作是一种最常用的乒乓操作方式，如图 2-2 为乒乓存储操作示意图。

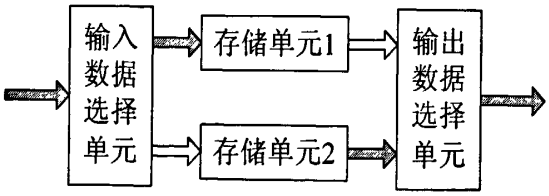

图 2-2 乒乓存储操作示意图[5]

通过输入数据流选择单元和输出数据流选择单元按节拍、相互配合的切换。整个乒乓存储模块对外只有一个读信号和一个写信号，读写相互配合，后级的模块在进行读操作时，不会与前级的写操作产生冲突。乒乓存储操作与流水线配合，可以保证流水线不会出现“断流”或者“溢出”的现象。

3. 串并转换

串并转换是 FPGA 设计的一个非常重要的思想，是 EDA 设计中经常用到的

数据流处理手段，从本质上来讲，串并转换实际上是设计中消耗的芯片面积与想要得到的设计工作速度的互换思想的体现。将串行转换为并行，一般旨在通过复制逻辑，提高整个设计的数据吞吐率，本质是通过芯片面积的消耗来换取系统工作频率的提高，而将并行转换为串行，则是在保证了工作频率的情况下尽量的节约芯片消耗的面积，以达到降低整个设计成本的目的。串并转换实现的方法多种多样，根据数据的排序和数量的要求可以选用寄存器、RAM 等实现。前面在乒乓操作的举例，就是通过 DPRAM 实现了数据流的串并转换，而且由于使用了 DPRAM，数据的缓冲区可以开的很大。对于数量比较小的设计可以采用寄存器完成串并转换。如无特殊需求，应该用同步时序设计完成串并之间的转换。对于排列顺序有规定的串并转换，可以用 case 语句判断实现。对于复杂的串并转换，还可以用状态机实现。

4. 模块化设计

模块化设计就是将设计按照流程、功能将系统划分为若干个模块，每一个模块也可以再划分为若干个子模块，结构层次化编码就是模块化设计思想的一种体现，目前大型设计中必须采用结构层次化编码风格，以提高代码的可读性，易于模块划分，易于分工协作，易于设计仿真测试激励。最基本的结构化层次是由一个顶层模块和若干个子模块构成，每个子模块根据需要还可以包含自己的子模块。合理的子模块划分不但有利于降低 EDA 综合布线工具的压力，还有利于进行团队合作，加快开发进度。

5. 模块复用

模块复用是从微观的角度来观察节省"面积"的问题。

在 FPGA 芯片中，许多资源是非常有限的，尤其是乘法器 DSP 资源。采用模块复用的方法将有助于节约这些资源，提高资源的利用率。下面举一个例子对模块复用进行讨论。

```
If(sel = 0)then          If(sel = 0)then
    F = A*B;                 mult1 = A;
Else                         mult2 = B;
    F = C*D;             Else
End if;                      mult1 = C;
                             mult2 = D;
                         End if;
                         F = mult1*mult2;
```

(a) (b)

图 2-3 模块复用示例[5]

在设计中，实现 F=[Sel ? (A×B) : (C×D)]运算时，如果直接按照其运算的方式编写代码（如图 2-3(a)所示）并综合实现，那么将需要用 2 个乘法器加一个双路选择器；而经过优化后，采用模块复用的方式，其代码如图 2-3(b)所示，经过

综合实现，只需要一个乘法器和两个双路选择器。由此可见，模块复用的方式可以使紧张的资源得到高效的利用，在实际应用中，这对于缓解 EDA 工具布局布线的压力也有很重要的作用。

2.4 StratixII 系列 FPGA 的特性和结构

2.4.1 StratixII 系列芯片的特点

Stratix II 是基于 1.2V、90nm 的全铜 FPGA 系列。Stratix II 采用新的逻辑结构，能够使性能达到最优，等效逻辑单元(LEs)达到 180000 个。Stratix II 系列器件提供多达 9 兆比特的片内三维矩阵型存储器，用来实现高存储要求的应用，和 96 个 DSP 块，可配置成 384(18-bit×18-bit)个乘法器，有效实现高性能滤波器和其他 DSP 功能。Stratix II 支持多种高速外围存储器接口，包括双倍速率(DDR)SDRAM 和第二代双倍速率(DDR2)SDRAM、第二代低延迟动态随机存取内存（RLDRAM）、第二代四倍速率（QDR)SDRAM 和单倍速率(SDR)SDRAM。Stratix II 芯片支持多种 I/O 标准，如具有动态相位调整电路的每秒 Gbit(Gbps)的源同步。Stratix II 提供完备的时钟管理方案，其内部时钟频率高达 550 MHz，内嵌 12 个锁相环（PLLs)。Stratix II 也是业界第一款使用先进编码标准(AES）算法翻译配置比特流的芯片，达到保护设计的目的[6]。

2.4.2 StratixII 芯片的结构

对于一般 FPGA 芯片，其逻辑单元(LE)经典结构为一个四输入查找表(LUT)和一个触发器。为什么几乎所有的 FPGA 厂商都采用这个结构呢？我们假设 FPGA 中有两种 LUT，一种是固定 2 输入的，另一种是固定 8 输入的，我们看一下这两种 LUT 如何实现一个 5 输入的函数。

在使用 2 输入的 LUT 时，4 个 LUT 串成一个链中。这时，设计中逻辑级数为 4 级，逻辑延时较大，但是所有的 LUT 输入都用上了。在使用 8 输入 LUT 时，只需要 1 个 LUT 时。这时，设计中逻辑级数为 1 级，逻辑比较快，但是有 3 个 LUT 的输入信号被浪费了，也就是浪费了部分硅片面积，如图 2-4 所示：

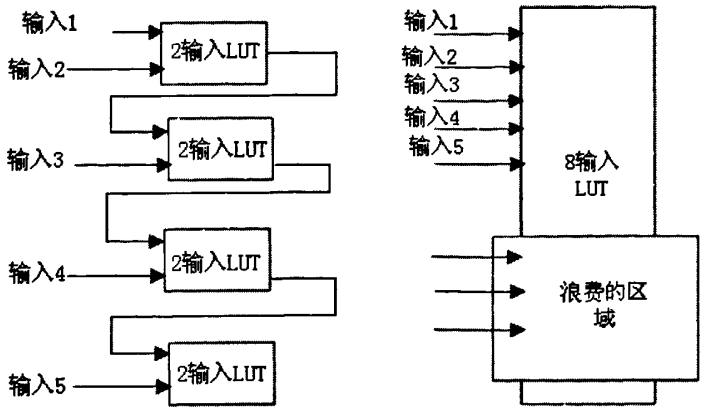

图 2-4 宽逻辑与窄逻辑比较示意图[5]

相比较而言，采用较"窄"的逻辑结构，比较节省硅片面积，但是总体性能比较差，而采用"宽"的逻辑结构，总体性能比较好，但是浪费硅片面积，成本较高。所以采用 4 输入的 LUT 只是在成本和性能之间做的一个折中。

Stratix2 的 ALM 正是兼有了"窄"的逻辑结构的高利用率和"宽"逻辑结构的高性能。ALM 中的组合逻辑模块可以根据用户的需求由设计工具自动配置成需要的模式。例如可以配置成 5 输入和 3 输入的 LUT，或 5 输入和 4 输入的 LUT，或者两个 4 输入的 LUT 等等，如图 2-5 所示：

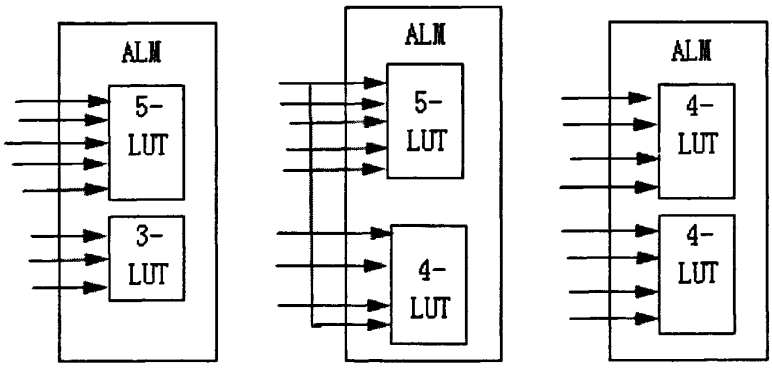

图 2-5 ALM 灵活配置示意图[5]

Stratix2 采用了 ALM 结构，可以说是 FPGA 构架方面的革命。它能在为用户提供高性能的同时，保持较低的成本。

另外，ALM 内部也有两个 3 输入加法器，和传统的 2 输入加法器相比，在实现算术运算时，显著的减少了加法电路的级数，提高了计算的性能。

第三章 核心比对模块的优化和改进

3.1 核心比对模块的总体框架

3.1.1 特征三角形比对

特征三角形比对模块采用流水线加并行处理的架构来实现，库指纹源源不断地送入该模块进行处理，最终每个现场指纹三角形最多得到 12 个匹配的库指纹三角形，对应最多 12 个偏移量[5]。根据算法的处理流程，整个模块可划分为 9 个子模块。即图 3-1 中的（1）~（9）。

通过大量的实验和测试，我们发现每个现场指纹可以用不多于 20 个现场指纹三角形来表示，所以我们的设计中的（3）特征边比对模块（4）边存储和（5）第三边输出模块采用了 20 路并行的结构，每一路对应一个现场三角形，分别独立并行地进行比对和存储。

图 3-1 是特征三角形比对的模块框图，图中黄色模块采用状态机来实现，蓝色模块采用流水线的结构。

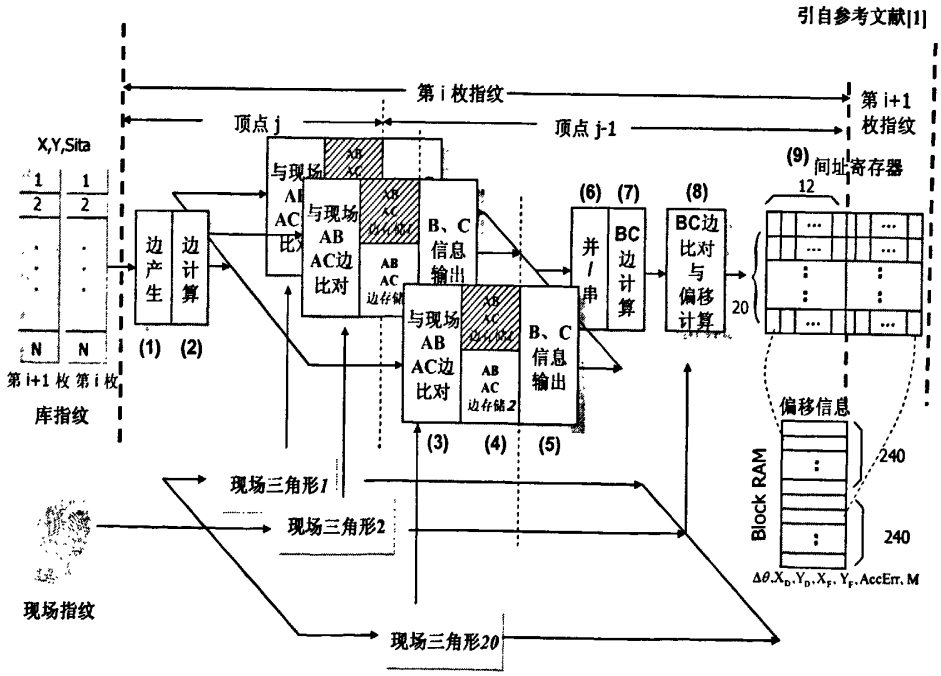

图 3-1 特征三角形比对模块设计结构示意图[5]

特征三角形的原始设计可参见参考文献【2】和参考文献【5】。

3.1.2 偏移量计算

偏移量计算模块在整体设计上采用了并行加流水线的设计方法，每枚库指纹与现场指纹比对之后得到的最多 240 个偏移量都需要经过此模块处理，最终得到最多 3 个能代表现场指纹和库指纹之间偏移的偏移量。图 3-2 是优化后的偏移量计算模块框图，整个模块可以划分为 7 个子模块，黄色的模块主要利用状态机完成，蓝色部分主要是采用流水线的设计方法完成。

引自参考文献[1]

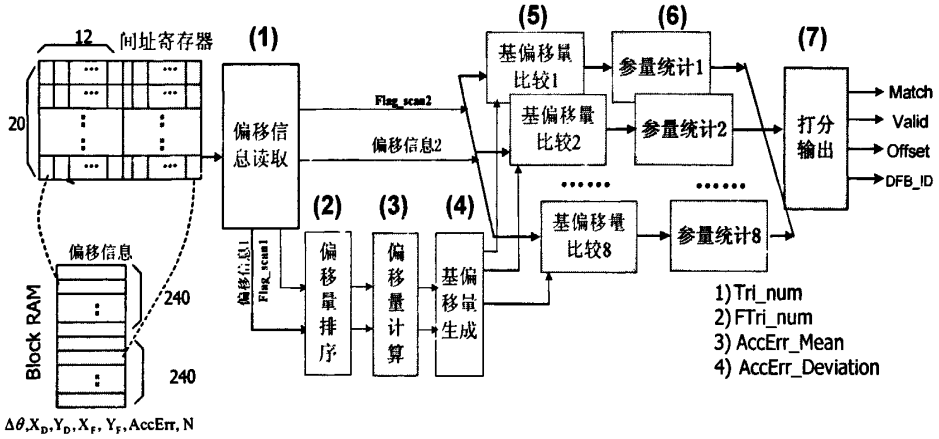

图 3-2 偏移量计算模块设计结构示意图[5]

偏移量计算的原始设计参见参考文献【2】和参考文献【5】。

3.1.3 细节点匹配

细节点匹配模块在实现时，采用了流水线、并行处理以及乒乓操作等设计方法，考虑到该模块与前面两个模块在运算速度上的匹配以及 FPGA 占用资源等问题，本模块的设计采用了半并行的处理方法。

本模块需要首先将现场指纹按照输入的偏移量进行旋转偏移，然后对每个现场指纹细节点找出与之匹配的库指纹细节点，在实现这一步骤时，需要采用并行处理的方法以减少运算时间，现场指纹最多是 80 个点，如果采用全并行的方式的话，就需要用 80 路并行，这显然是不现实的，占用资源过多。经过统计分析，有 60%的现场指纹其细节点数目在 32 以内，90%的现场指纹其细节点数目在 64 之内，细节点数更多的现场指纹比较罕见，如果采用 32 路的半并行比对，就可以在占用资源和运算时间这两个方面作一个比较妥善的折衷。

图 3-3 是细节点匹配的模块框图。

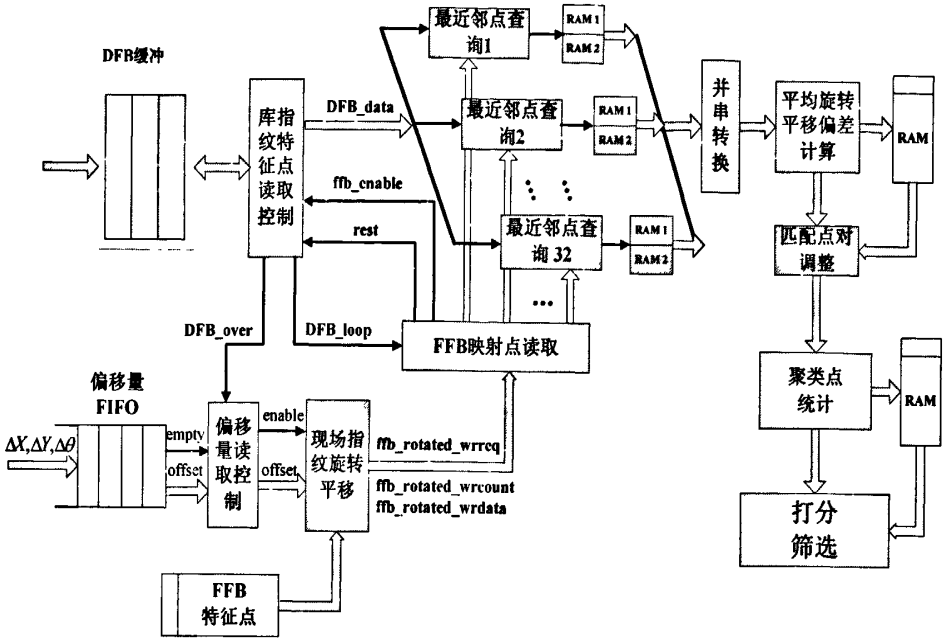

图 3-3 细节点匹配模块设计结构示意图

细节点匹配的原始设计参见参考文献【5】。

3.2 计数读取标识输出的边产生的改进

3.2.1 边产生模块的作用

边产生模块即图 3-3 的模块（1），为特征三角形比对模块提供源源不断的库指纹。边产生模块的片内库指纹缓冲采用乒乓操作。

3.2.2 原始设计的问题和相应的解决办法

（1） 输出当前库指纹的数据给 3 重乒乓 RAM

每一枚库指纹经过图 3-1 的特征三角形比对、图 3-2 的偏移量计算，最多生成 3 个有效基偏移量进入图 3-3 的细节点匹配模块。在细节点匹配过程中，图 3-3 的库指纹特征点读取控制子模块需要重新读取该库指纹。由于整个系统基于流水线架构，此时该库指纹已不在边产生模块的片内库指纹缓冲里面了。因此需要另设计一个缓冲，即图 3-3 中的 DFB 缓冲，来暂存该库指纹，供细节点匹配模块使用。DFB 缓冲的设计在 3-5 节介绍。

边产生模块以二重循环方式工作，外循环是所有特征点遍历一遍，对应外循环的每一个特征点，内循环再将所有特征点遍历一遍。因此可在外循环过程中将特征点写入图 3-3 中的 DFB 缓冲。这样做的好处是可以直接使用外循环点的有效信号和地址。

(2) 引入 matchsuspend 信号

整个核心比对模块基于流水线架构，最后的细节点匹配模块将匹配结果，即可能与现场指纹匹配的库指纹序号写到一个 FIFO 里，待到 FIFO 满到一定程度就由 FIFO 写控制模块内部中断信号，指示主机读取匹配库指纹 ID。该内部中断经本地总线接口模块 local_9054 反映到本地总线的 LINT#上。9054 收到这个中断后，再转化为 PCI 总线上的中断通知主机，由主机发命令读取匹配结果。由于主机这时可能忙于其它工作，从 FIFO 写控制模块发送内部中断到读取匹配结果有一个延时，为提高比对速率，这段时间内流水线不能断，所以不能等到 FIFO 满了才发内部中断，必须在 FIFO 里留出一定的裕量存放这段时间内可能产生的匹配库指纹 ID。

即使留出裕量，如果从 FIFO 写控制模块发送内部中断到读取匹配结果的延时很大， FIFO 完全充满的危险依然存在，后续的写操作就会引起数据的损毁或丢失。因此需要设计保护措施，在 FIFO 将要充满时中止核心比对模块的工作。

核心比对模块以流水线方式工作。特征三角形比对处理 1 枚库指纹时，偏移量计算模块处理之前的 1 枚指纹，细节点匹配模块正在处理更前 1 枚指纹。这 3 个模块之间通过握手信号实现数据传递。理论上打断流水线的任何一级都可以暂停流水线，但从本设计的特点考虑，假设在流水线的中间部分截断流水线，由于前面的模块还在继续工作，不设缓冲势必导致数据的溢出。因此应尽量在靠近数据流的源头出截断流水线。边产生模块位于整个流水线的前端，因此选择对它进行修改。

新的边产生模块在每次结束对一枚库指纹的二重循环遍历后判断从库指纹 FIFO 控制器传过来的信号 matchsuspend，如果 matchsuspend 为'1'，则进入挂起状态直到 matchsuspend 释放。

3.2.3 改进后的特征边生成模块

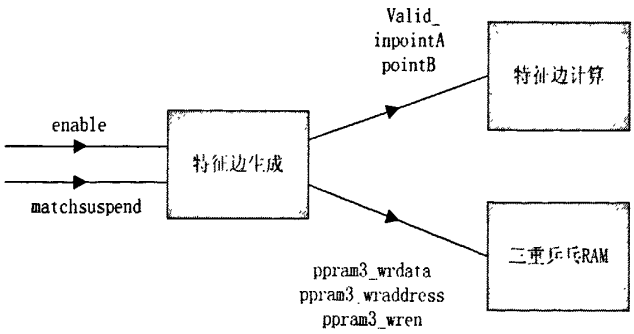

图 3-4 特征边生成模块数据流图

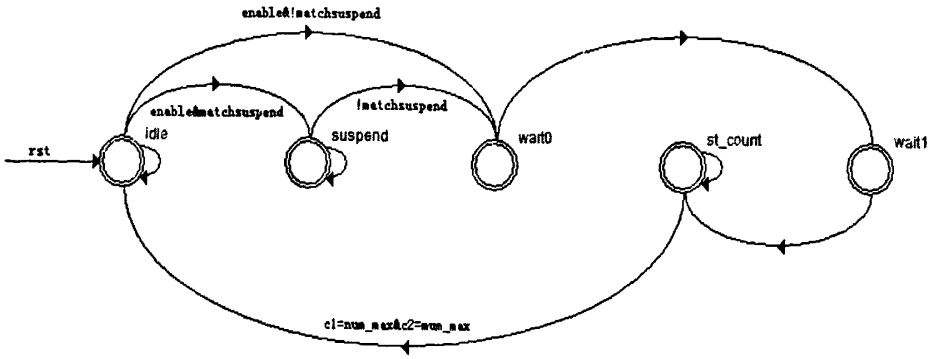

图 3-5 改进后的边产生状态机

如图 3-5 所示，系统复位后进入 idle 状态。enable='1'表示片内库指纹缓冲乒乓控制器已经把库指纹准备好了，这时如果 matchsuspend 为'1'，说明后面的 FIFO 即将被充满，因此进入 suspend 状态，反之直接进入 wait0。在 suspend 判断 matchsuspend，matchsuspend 变成'0'表示 FIFO 已经倒出足够的空间，跳到 wait0，再过 1 个时钟跳到 wait1，读取第一个外循环点。之后进入 st_count，在 st_count 进行二重循环，内层循环的指针是 c2,外层循环的指针是 c1，它们被转化成输入到片内乒乓 RAM 的地址 addr,有效的 c1 和 c2 的组合表示一条特征边，二重循环也是特征边的生成过程。c2 从 0 递增到 num_max，完成一次内循环，c1 自增 1。c1 和 c2 都等于 num_max 表示二重循环结束，状态机回到 idle，准备读取下一枚库指纹。特征边的生成如图 3-6 所示。

当前库指纹的特征点的数目存放在片内乒乓 RAM 之一的首地址，在 wait0 状态读取它，两个时钟后读出库指纹特征点的数目 num_max 和库指纹序号 FID，并有效 FID_en，传给 matchover_gen，matchover_gen 根据 FID 和 FID_en 产生比对结束信号 matchover，matchover_gen 的作用后面会有详细说明。

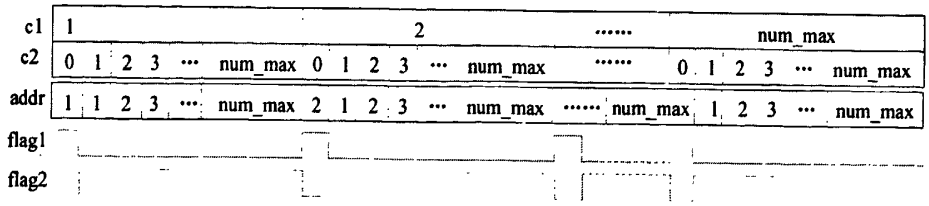

图 3-6 特征边生成示意图[5]

在图 3-6 中， flag1 有效时表示读取外循环点，把它延时几个周期后作为写使能信号，和读出来的外循环特征点等数据一起传给三重乒乓 RAM。

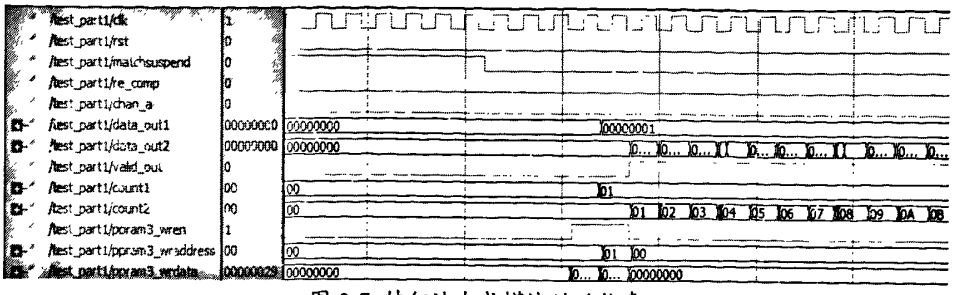

图 3-7 特征边生成模块的后仿真

图 3-7 中，matchsuspend 为'1'，表示 FIFO 的写控制模块要求中止比对，边生成模块处理完当前库指纹后暂停，直到 matchsuspend='0'才继续工作。库指纹前两个数据也发给三重乒乓 RAM。

3.3 特征三角形比对模块中并串转换模块的改进

3.3.1 进行并串转换的原因

特征三角形比对的并串转换子模块即图 3-1 的第（6）部分。算法中将现场三角形中的特征点组合成最多 20 个现场三角形，为了加快比对速度，采用 20 路并行比对的结构。每一路统计出与现场三角形 ABC 的 AB 边和 AC 边最匹配的若干条库指纹特征边 A`B`和 A`C`，并两两组合成 B`C`，发送到后级模块进行与现场指纹特征边 BC 的比对。由于经过 AB、AC 边比对模块后，大多数特征边被淘汰，因此在 B`C`边输出端口和与现场三角形的 BC 边比对电路之间加入并串转换模块，这样可在保证流水线数据速率匹配的前提下减小综合后的面积。

该模块主要输入信号有：时钟信号 clk、复位信号 rst、require[19:0]信号（由 20 路第三边产生模块中 require 信号组成，指示各路是否需要输出第三边）、valid[19:0]信号(由 20 路第三边产生模块中的输出有效信号组成，用来指示输出的第三边信息是否有效)、data_in（由第三边产生模块输出的第三边信息）、re_comp（库指纹切换信号）、require_valid（外循环点切换信号）。

主要的输出信号：valid（指示输出的经过串行变换后的数据是否有效）、ack[19:0]（输出到上级模块的响应信号，用来控制上级并行模块使其串行输出第三边信息）、N_out（用来指示输出的第三边数据来自哪一路，即该第三边需要和哪个现场三角形进行比较）、data_out（经过串行变换后输出的第三边信息）、re_comp（库指纹切换信号）。

从收到 require[19:0]到开始响应，这中间插入了 3 级中间处理单元。第一级将 require[19:0]分成 4 组，分别是 require[19:15]、require[14:10]、require[9:5]、require[4:0]，将每一组的 5 路请求信号转换到 temp_index 中去。以 require[4:0]

为例，把 require[4:0]中最多 5 路有效的请求信号转存到 temp_index[4:0]里，temp_index[4:0]中，序号小的优先。也就是说，在 temp_index[i-1:0]填满之前，不允许使用 temp_index[i]。第二级和第三级与之类似，这样经过 3 级变换单元后，require[19:0]中分散的请求信号被映射到 temp_index[19:0] 连续存储，并设置 total 记录总的请求数目，然后设立游标 nn，从 temp_index[0]开始依次响应 temp_index[nn]指代的请求，循环 total 次后结束。

这样做符合前文提到的流水线设计方法，因为直接判断 20 路 require 信号并依次给出对应的 ack 的逻辑很复杂，延时很可能不满足 66M 的系统时钟的要求。分成 3 级小逻辑，就可以保证每一级都能工作在 66M 时钟下。

虽然经过论证，前级处理单元发过来的有效数据很少，但仍然有溢出的风险，因此原始算法在 require[19:0]中不会有太多请求的前提下，限定了其中每一个请求有效 B`C`的数量，使得在外循环点两次切换的间隔内能够完成对 requre[19:0]的一次扫描。

3.3.2 原始并串转换模块的隐患

原始算法的隐患在于 3 级映射电路的引入，使 require_valid 也必须相应插入 3 个时钟周期的延时，即并串转换内部用到的外循环点切换信号 enable3 是 require_valid 经过 3 级触发器后的信号。然而对 re_comp_5_6 却没做相应的延时，后果是，输入的有效的 require_valid 和有效的 re_comp_5_6 是对齐的，而在并串转换模块内部，re_comp_5_6 领先于最后的 enable3 两个时钟周期变化，破坏了它们原有的同步关系。

更严重的是，原有电路在外循环点切换时开始以上一个外循环点为顶点的 A`B`、A`C`边的并串转换。当 re_comp_5_6 有效时，对应最后一个外循环点的数据还没经过并串转换。特征三角形比对和偏移量计算之间用一组乒乓存储器存放中间数据，经过若干级触发器后，re_comp_5_6 成为该乒乓存储器的切换信号，特征三角形比对模块内部是流水线结构，这样乒乓存储器的切换也将早于最后一个外循环点的有效数据，最后一个外循环点的有效数据将被作为下一枚指纹的数据来处理，造成错位。

3.3.3 对并串转换模块的改进

针对这个问题，采取“先暂存，再释放”的办法。

因为 re_comp_5_6 与最后一个外循环点对应的 require_valid 同时有效，从 require_valid 有效到 total 有效，要经过 3 个时钟周期，因此将 re_comp_5_6 也延时 3 个时钟周期，得到与 total 同步的 temp_re_comp3。

temp_re_comp3 如果是`1`，先看 total 是否是'0'，total 不为'0'表示后面还有有效数据，就先将 temp_re_comp3 暂存入 temp_re_comp。等到处理完这批有效数据再把 temp_re_comp 赋给 re_comp_6_7。如果 total 是'0'直接置 re_comp_6_7 为`1`。

如果 temp_re_comp3 是`0`，说明最后一个外循环点还没到。置 re_comp_6_7 为`0`。

用状态机模型可以表述如图 3-8：

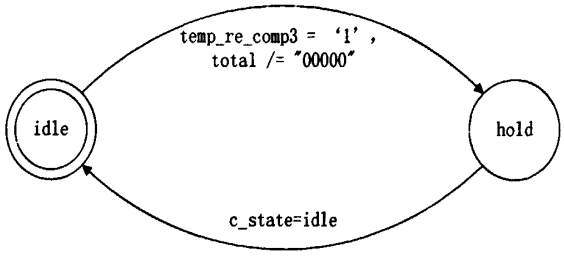

图 3-8 产生 re_comp_6_7 的状态机 r_state

c_state 是程序主状态机，在 r_state 处于暂存状态 hold 的同时，c_state 也进入回复 require 的 acknowledge 和 read，当 c_state 回到 idle，意味着以最后一个外循环点为顶点的第三边数据被处理完。这时候 r_state 回到 idle，同时把 re_comp_6_7 置为高电平。

改进后程序的后仿真波形如图 3-9 所示：

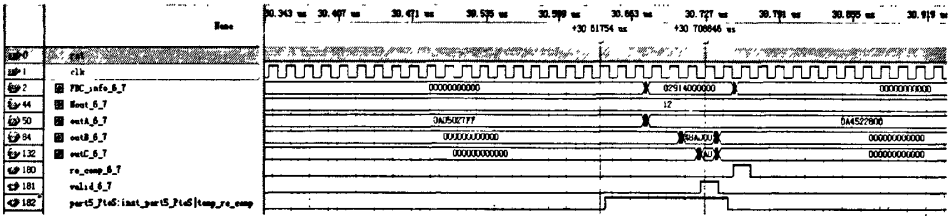

图 3-9 改进后的 re_comp_6_7 的波形

从图 3-9 可见，在第二个标尺处是以最后一个外循环点为顶点的第三边的数据，temp_re_comp 在之前被置为高电平，一直到这批数据被输出到下一级，re_comp_6_7 才被赋为'1'，同时 temp_re_comp 归零。

作为对比，将改进前的程序的后仿真波形列于图 3-10

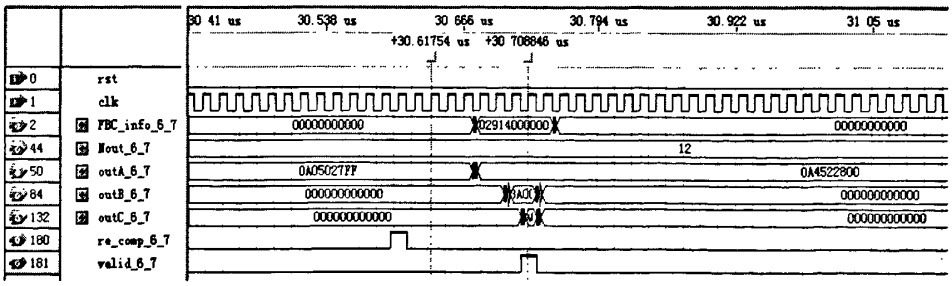

图 3-10 改进前的 re_comp_6_7 的波形

图 3-10 中，re_comp_6_7 输出高电平时，最后一组有效数据还在处理中，表现为对应 outA_6_7 的值是"0A4522800"的那组数据在 re_comp_6_7 有效后才输出，相应地，在图 3-6 中，这组数据输出后，re_comp_6_7 才有效。可见改进后的效果。

3.4 对偏移量排序模块的改进

3.4.1 偏移量排序模块的功能

偏移量排序模块是图 3-2 的第（2）部分。在经过特征三角形比对后，生成的偏移信息按照其对应的现场三角形排列存储，对每一个现场三角形最多存储 12 个偏移信息，且这 12 个偏移信息是按照累积误差从小到大的顺序排列的。生成基偏移量时，需要进行第一次偏移信息读取，即先依次读出每个现场三角形（共 20 个）的当前的累积误差最小的偏移信息，再读第二小的，以此类推，送入偏移量计算以及后级的基偏移量生成模块。如图 3-11 所示。

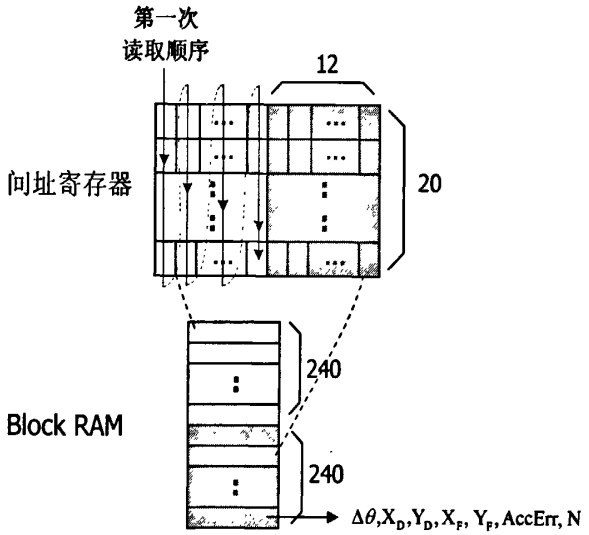

图 3-11 偏移信息第一次读取顺序示意图[21]

虽然每次读出的偏移信息的累积误差都是每个现场三角形中现存最小的，但是这 20 个偏移信息之间的累积误差却是无序的。也就是说，图 3-11 中，间址寄存器阵列中，每一行指向的偏移信息的累积误差是从小到大的，但每一列指向的偏移信息的累积误差却是无序的。如果累积误差大的偏移信息先进入基偏移量生成模块并成为一个基偏移量，那么排在后面的累积误差较小的偏移信息就可能被淘汰，导致最终生成的基偏移量不是最优的。因此有必要在偏移信息读取模块和偏移量计算模块之间插入偏移量排序模块，对输入的偏移信息按照累积误差从小到大进行排序。

偏移量排序模块的输入信号有时钟 clk；复位信号 rst；对偏移信息进行第一次读取的标志信号 flag_in；every20_in（间址寄存器每一列最后一个偏移信息输出时该信号有效）；标志偏移信息有效的信号 valid_in；offset_in——偏移信息，包括 DX、DY、FX、FY、delta、AccErr、 stop，当 8 个偏移量都产生后，由后级的偏移量生成模块置为高电平，通知偏移量排序模块复位。

偏移量排序模块的输出信号有 flag_out，有效时表示进行偏移信息的第一次遍历；every20_out（间址寄存器每一列最后一个偏移信息输出时该信号有效）；valid_out，标志偏移信息有效；offset_out——按列排序后的输出到后级的偏移信息。

3.4.2 原设计存在的问题

原始设计主要存在以下问题：

1. 原始设计的偏移量排序模块中，先检测 flag_in，如果 flag_in 是'1'再看 every_20 的值，如果这时 every_20 也是'1'，就切换乒乓 RAM 并输出排好序的偏移信息给后级的偏移量生成模块。每一个现场三角形最多有 12 个偏移信息，flag_in 的下降沿和 every20_in 的第 12 个上升沿同时发生，导致最后一列偏移信息没有机会参与基偏移量的统计。

2. 在每个时钟的上升沿，如果检测到 every_20 是'1'，就进行乒乓 RAM 的切换，把上一列的有效偏移信息数目 num1 赋给 num2，同时置 read_enable 为'1'，一个时钟周期后开始读取暂存在乒乓 RAM 的上一列偏移信息，读取办法是用一个计数器 count 从 0 计数到 num2，num2 最大值是 20，所以最多要用 21 个时钟周期结束一次遍历。而 every20_in 每 20 个时钟有效一次，意味着 num2 每隔 20 个时钟更新。如果上一列有 20 个有效偏移信息，当前列的有效偏移信息不足 20 个，那么 num2 会先于 count 更新，当 count 值是 20，num2 的值小于 20，count 就会继续累加，状态机不仅不能及时退出读取上一列偏移信息的循环，还会造成对当前列偏移信息读取的错位。

3. 原设计的状态机用一段式撰写，可读性较差，不利于综合器优化。

综上所述，重新编写偏移量排序模块。

3.4.3 新的偏移量排序模块

原始设计中的偏移量生成模块要求有效的 every20 和当前列的最后一个偏移信息同时输出，而原设计为了便于排序，将 every20_in 多加了一个时钟的延时，将有效的 every20_in 与当前列的第一个偏移信息同时输出，由偏移量排序模块完成 every20_in 的重新定位。这样一来偏移量排序模块的正常工作以前级模块的"错误"时序为前提，给调试和阅读带来困难。

因此首先修改前一级的偏移量遍历模块，去掉寄存 every20_in 的流水线。

为简化设计，将偏移量排序模块分为排序模块和读偏移量状态机。排序模块完成排序算法，偏移量读取状态机把排好序的偏移量交给下一级。

与软件排序不同，排序模块在接收偏移量的同时完成排序，方法是在系统复位后将辅助的 mem_accerr 置为全 1，num1 清零，mem_accerr 是一个 20 维的向量，mem_accerr(i)存放第 i 个偏移量的累积误差，num1 记录每列有效的偏移量个数。

排序是将累积误差按升序排列的过程。flag_in='1'时开始排序， valid_in='1'表示输入的偏移量有效，排序算法用行为级描述是：将该偏移量的累积误差首先和 mem_accerr(0)比较，如果该小于 mem_accerr(0)，那么把 mem_accerr（18）至 mem_accerr(0)依次后移，然后用 mem_accerr(0)存放该偏移量的累积误差；否则就和 mem_accerr(1)，mem_accerr（2）.....比较，直到找到存放该偏移量的累积误差的位置。然后 num1 加 1，由于需要暂存的数据共有 2×20×12×55=26400 个比特之多，用 QuartusII 例化 lpm_ram 的 IP 核，使用芯片内部的 RAM 存放该偏移量，另分配一个 20 维的向量 mem_address1，mem_address1(i)是第 i 个偏移量在 RAM 中的位置。

every20_in='1'表示当前的偏移量是本列最后一个偏移量，将 mem_address1、num1 分别赋给 mem_address2 和 num2，切换乒乓 RAM，同时有效 read_enable，最后把 mem_address1 置为最大值，即全 1，把 num1 清零，准备对下一列排序。

为提高可读性和便于综合器的优化，读偏移量状态机的设计采用两段式，如图 3-12 所示：

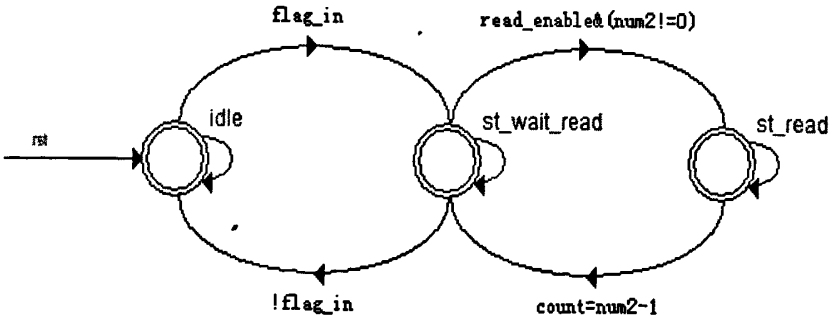

图 3-12 读偏移量状态机

系统复位后状态机处于 idle 状态，在 idle 状态读计数器 count 和读 RAM 使能信号都是 0，检测到 flag_in='1'后，进入等待状态 st_wait_read。在 st_wait_read 如果检测到 read_enable 为'1'，且 num2 不为 0，那么状态机进入 st_read；如果 num2 是 0，表明上一列没有有效偏移量，也就没有数据发给后级，状态机停在 st_wait_read。

考虑到在收到有效的 read_enable 的同时，偏移量排序模块实际已将上一列最后一个偏移量写到乒乓 RAM 里了，所以在 st_wait_read 状态用 read_enable 作为 RAM 的读使能信号 rden,读计数器 count 之前是 0,在 st_wait_read 转向 st_read 的同时自增 1，这样在 st_wait_read 转向 st_read 时读第一个偏移量。在 st_read 状态置 RAM 读信号 rden 为'1'，count 从 1 计到 num2-1，把 mem_address2(count) 指向的偏移量发给后级，随后返回 st_wait_read。

每一列最多有 20 个有效偏移量，即 num2 最大值是 20。在读上一列第一个有效偏移量的同时，当前列的第一个偏移量正在进行排序，再过 19 个时钟周期后 num2 被更新，count 最迟也能在 num2 更新前一个时钟沿累计到 num2-1（18 个时钟周期后），硬件的并发性质保证了状态机最迟也能在 num2 更新的同时回到 st_wait_read，一个时钟后读取当前列的偏移量。上文提到的问题（2）得到了解决。

在 st_wait_read 如果 flag_in 无效，说明第一次偏移量遍历结束，状态机回到 idle。

选择时钟输入管脚为 Pin_U3,复位输入管脚为 Pin_A6，其他管脚为虚拟管脚，禁止优化上电电平，闲置管脚设为"As input tri-stated with bus-hold circuitry"，其他综合选项按默认设置。选择 Quartus II 9.1 开发环境，改进后的与原始的全编译报告和前 10 条最大延时路径报告如下：

Flow Status	Successful - Sat Jan 16 16:24:27 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	part9_sort
Top-level Entity Name	part9_sort
Family	Stratix II
Device	EP2S90F1020C5
Timing Models	Final
Met timing requirements	Yes
Logic utilization	1 %
Combinational ALUTs	535 / 72,768 (< 1 %)
Dedicated logic registers	357 / 72,768 (< 1 %)
Total registers	357
Total pins	2 / 759 (< 1 %)
Total virtual pins	116
Total block memory bits	3,520 / 4,520,448 (< 1 %)
DSP block 9-bit elements	0 / 384 (0 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

图 3-13 改进后的偏移量排序模块全编译报告

Flow Status	Successful - Wed Jan 13 15:07:55 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	part9_sort
Top-level Entity Name	part9_sort
Family	Stratix II
Device	EP2S90F1020C5
Timing Models	Final
Met timing requirements	Yes
Logic utilization	1 %
Combinational ALUTs	447 / 72,768 (< 1 %)
Dedicated logic registers	430 / 72,768 (< 1 %)
Total registers	430
Total pins	2 / 759 (< 1 %)
Total virtual pins	117
Total block memory bits	3,520 / 4,520,448 (< 1 %)
DSP block 9-bit elements	0 / 384 (0 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

图 3-14 原始的偏移量排序模块全编译报告

	Slack	Actual fmax (period)	From	To	From Clock	To Clock	Required Setup Relationship	Required Longest P2P Time	Actual Longest P2P Time
1	N/A	141.74 MHz (period = 7.055 ns)	mem_accen[1][0]	mem_accen[16][1]	clk	clk	None	None	6.719 ns
2	N/A	141.74 MHz (period = 7.055 ns)	mem_accen[1][0]	mem_accen[16][2]	clk	clk	None	None	6.719 ns
3	N/A	142.43 MHz (period = 7.021 ns)	mem_accen[2][5]	mem_accen[16][1]	clk	clk	None	None	6.685 ns
4	N/A	142.43 MHz (period = 7.021 ns)	mem_accen[2][5]	mem_accen[16][2]	clk	clk	None	None	6.685 ns
5	N/A	143.04 MHz (period = 6.991 ns)	mem_accen[1][2]	mem_accen[16][1]	clk	clk	None	None	6.655 ns
6	N/A	143.04 MHz (period = 6.991 ns)	mem_accen[1][2]	mem_accen[16][2]	clk	clk	None	None	6.655 ns
7	N/A	143.88 MHz (period = 6.950 ns)	mem_accen[1][5]	mem_accen[16][1]	clk	clk	None	None	6.682 ns
8	N/A	143.88 MHz (period = 6.950 ns)	mem_accen[1][5]	mem_accen[16][2]	clk	clk	None	None	6.682 ns
9	N/A	144.51 MHz (period = 6.920 ns)	mem_accen[9][6]	mem_accen[16][1]	clk	clk	None	None	6.573 ns
10	N/A	144.51 MHz (period = 6.920 ns)	mem_accen[9][6]	mem_accen[16][2]	clk	clk	None	None	6.573 ns

图 3-15 改进后的偏移量排序模块的前 10 条最长延时路径

	Slack	Actual fmax (period)	From	To	From Clock	To Clock	Required Setup Relationship	Required Longest P2P Time	Actual Longest P2P Time
1	N/A	128.21 MHz (period = 7.800 ns)	mem_accen[4][4]	mem_address1[18][0]	clk	clk	None	None	7.550 ns
2	N/A	128.40 MHz (period = 7.788 ns)	mem_accen[4][4]	mem_accen[18][2]	clk	clk	None	None	7.538 ns
3	N/A	128.40 MHz (period = 7.788 ns)	mem_accen[4][4]	mem_accen[18][0]	clk	clk	None	None	7.538 ns
4	N/A	128.40 MHz (period = 7.788 ns)	mem_accen[4][4]	mem_accen[18][1]	clk	clk	None	None	7.538 ns
5	N/A	128.78 MHz (period = 7.765 ns)	mem_accen[4][4]	mem_accen[18][5]	clk	clk	None	None	7.515 ns
6	N/A	128.78 MHz (period = 7.765 ns)	mem_accen[4][4]	mem_accen[18][4]	clk	clk	None	None	7.515 ns
7	N/A	129.53 MHz (period = 7.720 ns)	mem_accen[1][4]	mem_address1[18][0]	clk	clk	None	None	7.465 ns
8	N/A	129.74 MHz (period = 7.708 ns)	mem_accen[1][4]	mem_accen[18][2]	clk	clk	None	None	7.453 ns
9	N/A	129.74 MHz (period = 7.708 ns)	mem_accen[1][4]	mem_accen[18][0]	clk	clk	None	None	7.453 ns
10	N/A	129.74 MHz (period = 7.708 ns)	mem_accen[1][4]	mem_accen[18][1]	clk	clk	None	None	7.453 ns

图 3-16 原始的偏移量排序模块的前 10 条最长延时路径

从图 3-13 到图 3-16 可见，改进后的偏移量排序模块比原设计多用了 88 个 ALUT，少用了 73 个寄存器，可以运行的最高频率提高了 13.53MHz，考虑到节省了前级的偏移量遍历模块寄存 every20_in 的一个寄存器，还有改进后的偏移量排序模块的响应变快，而且上文提到的几个问题也都得到了解决，新的偏移量排序模块基本满足要求。

改进后的排序模块的时序仿真波形如下图所示：

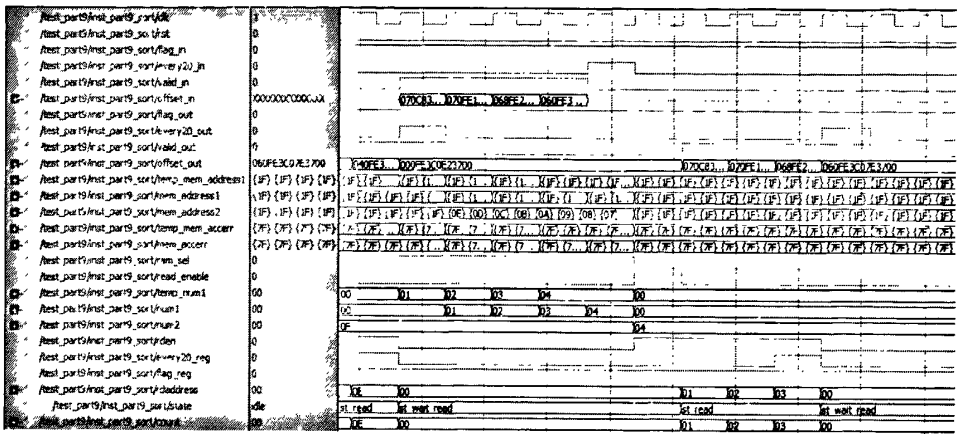

图 3-17 改进后的排序模块的功能仿真波形

3.5 对细节点匹配模块的 DFB 缓冲的修改

3.5.1 细节点匹配模块的 DFB 缓冲的作用

图 3-3 中，现场指纹首先根据每个基偏移量映射到库指纹坐标系里，FFB 映射点读取模块把映射点数据传送到 32 路并行比对模块之后，将 ffb_enable 置为'1'，指示库指纹特征点读取模块给出库指纹数据。由于系统基于流水线架构，所以需要用乒乓 RAM 组成 DFB 缓冲，暂存库指纹。库指纹特征点读取控制模块在 ffb_enable='1'时，从 DFB 缓冲里读出库指纹，送到 32 路并行的最近邻点查询模块。

3.5.2 原始设计的 DFB 缓冲存在的问题

原始设计的 DFB 缓冲由四重乒乓 RAM 组成，它的读指针和写指针都在特征三角形比对模块的边生成模块完成遍历后更新，写指针始终领先读指针一个指纹，写指针始终指向将要从片内库指纹缓冲中读的库指纹，读指针始终指向正在特征三角形比对模块进行比对的那枚库指纹。

特征三角形比对模块的边生成模块完成遍历后，一枚库指纹经过边计算、并串转换、第三边比对、偏移量计算，到达细节点匹配模块。细节点匹配模块的库指纹特征点读取控制模块最多要读 3 次库指纹，这个步骤尤其耗时，在特征三角形比对模块的边生成模块完成遍历后和细节点匹配模块的库指纹特征点读取控制模块第 3 次读库指纹这段时间，读指针很有可能又经历了更新，读出来的不再是需要的库指纹。

另外，原设计没有设计写和读乒乓 RAM 的过程。综上考虑，重新编写细节点匹配模块的乒乓 RAM。

3.5.3 新的 DFB 缓冲

为了减小系统开销和降低设计难度，应该把写乒乓 RAM 和读乒乓的时刻尽量挨得近一些。在边产生模块的遍历过程中，在读每个库指纹的起始地址时，顺便也把起始地址的内容——库指纹 ID 和特征点数目也写到乒乓 RAM 里，每读一个外循环点也随即写到乒乓 RAM 里。这样伴随着边产生模块的外循环点循环，一枚完整的库指纹也被写到了细节点匹配的乒乓 RAM 里。二重循环结束后，有效库指纹切换信号 re_comp，连到乒乓 RAM 作为写指针和读指针的更新信号。

边产生模块循环，同时也是多重乒乓 RAM 指针更新的周期是

$$d(d+1)+2 \qquad (3\text{-}1)$$

d 是库指纹特征点数。

从边产生模块发给边计算模块的指纹切换信号 re_comp_1_2 有效到 BC 边比对与偏移计算模块输出库指纹切换信号，经过的延时是

$$d+21 \qquad (3\text{-}2)$$

d 是库指纹特征点数。

从 BC 边比对与偏移计算模块输出库指纹切换信号，到偏移量计算模块输出第一个有效基偏移量，经过的延时是

$$2\sum_{i=1}^{20} x_i + x_{20} + 322 + n + K(x_{20}) + \max(temp\_Tnum_j) \qquad (3\text{-}3)$$

n 是现场指纹特征三角形数，x_n 是第 i 个特征三角形的偏移信息数目，[redacted] 时，$K(x_{20})=1$，$x_{20}\neq 0$ 时，$K(x_{20})=0$，$temp\_Tnum_j$ 是落入第 j 个基偏移量的偏移信息数目，$1\le j\le 8, 0\le temp\_Tnum_j\le 64$。

从 BC 边比对与偏移计算模块输出库指纹切换信号，到偏移量完成第 2 次遍历，经过的延时是

$$2\sum_{i=1}^{20} x_i + x_{20} + 293 + n + K(x_{20}) \qquad (3\text{-}4)$$

设现场指纹特征点数目多于 32 个，那么从偏移量计算模块输出第一个有效基偏移量到细节点匹配模块完成对第一个基偏移量的计算（以 offset_read_valid='1'为标志），经过的延时是

$$93 + (d+2)\lceil f/32 \rceil + \text{ptos\_count\_match} \qquad (3\text{-}5)$$

其中，d 是库指纹特征点数，f 是现场指纹特征点数，ptos_count_match 是匹配点对总数。

设现场指纹特征点数目少于33个，多于16个，那么从偏移量计算模块输出第一个有效基偏移量到细节匹配模块完成对第一个基偏移量的计算，经过的延时是

$$f+d+63+ptos\_count\_match \tag{3-6}$$

设现场指纹特征点数目少于17个，那么从偏移量计算模块输出第一个有效基偏移量到细节点匹配模块完成对第一个基偏移量的计算，经过的延时是

$$\begin{aligned} &ptos\_count\_match+f+d+60+1+2+m\_realMNum^2-L(m\_realMnum)= \\ &f+d+ptos\_count\_match+m\_realMNum^2+63-L(m\_realMNum) \end{aligned} \tag{3-7}$$

ptos_count_match+f+d+60+1是聚类点统计固定延时，m_realMNum是匹配点对数目，m_realMNum<17，m_realMNum=0 或 1 时，L(m_realMNum)=1，m_realMNum=其他值时，L(m_realMNum)=0。

公式（3-5）~（3-7）中，除了现场指纹特征点数目f是常量，其它参数受库指纹的影响。

BC边比对与偏移计算输出前后两枚库指纹切换信号re_comp_8_9的间隔是

$$d_2(d_2+1)+2+d_2-d_1 \tag{3-8}$$

其中，d_2是当前库指纹的特征点数目，d_1是上一枚库指纹特征点数目。

20路现场特征三角形的乒乓RAM把排好序的偏移信息输入大的BLOCK RAM用的时间是

$$42+\sum_{i=1}^{20}x_i+n \tag{3-9}$$

公式（3-8）的最小值是40×（40+1）+2+40-127=1555

公式（3-4）的最大值是62+240=302

(3-8)恒大于(3-4)，流水线不会堵塞。

20路现场特征三角形的乒乓RAM前后两次输出ready20的时间间隔是

$$d_2(d_2+1)+2+d_2-d_1+(\sum_{i=1}^{20}x_{2i}-\sum_{i=1}^{20}x_{1i})+(n_2-n_1) \tag{3-10}$$

偏移量计算模块完成对BLOCK RAM的两次遍历的时间间隔是

$$\sum_{i=1}^{20}x_i+x_{20}+K(x_{20})+251 \tag{3-11}$$

公式（3-10）的最小值是40×（40+1）+2+40-127+(1-240)+(1-20)=1297

公式（3-11）的最大值是240+12+251=503

(3-10)恒大于(3-11)，流水线不会堵塞。

由(3-2)和（3-3）得，从边产生模块发给边计算模块的指纹切换信号re_comp_1_2有效到偏移量输出模块输出第一个有效基偏移量，经过的延时是

$$2\sum_{i=1}^{20}x_i+x_{20}+343+n+K(x_{20})+\max(temp\_Tnum_j)+d \tag{3-12}$$

由（3-1）和（3-12）得，偏移量输出模块输出前后两枚库指纹的第一个有效基偏移量的间隔是

$$d_2(d_2+1)+2+(d_2-d_1)+2(\sum_{i=1}^{20}x_{2,i}-\sum_{i=1}^{20}x_{1,i})+(x_{2,20}-x_{1,20})+(n_2-n_1)+ \\ K(x_{2,20})-K(x_{1,20})+\max(temp\_Tnum_{2,j})-\max(temp\_Tnum_{1,j}) \tag{3-13}$$

要使流水线不发生堵塞，细节点匹配模块必须在（3-13）表示的间隔内完成对一枚库指纹所有有效基偏移量的处理。

比较公式（3-5）、（3-6）、（3-7），当d恒定不变时，

$$(3-5)_{\min}=(3-5)|_{f=33,ptos\_count\_match=0}=2d+97,$$
$$(3-6)_{\max}=(3-6)|_{f=32,ptos\_count\_match=32}=d+127,$$
$$(3-5)_{\min}>(3-6)_{\max}$$

$$(3-5)_{\max}-(3-7)_{\max}= \\ (3-5)|_{f=ptos\_count\_match=80}-(3-7)|_{f=ptos\_count\_matchm\_realMNum=16}=2d-165$$

$d\geq 83$时，$(3-5)_{\max}>(3-7)_{\max}$，$d<83$时，$(3-5)_{\max}<(3-7)_{\max}$。

由（3-13）得，偏移量输出模块输出前后t枚指纹的第一个有效基偏移量的间隔是

$$\sum_{i=2}^{t}d_i(d_i+1)+2(t-1)+(d_t-d_1)+2(\sum_{i=1}^{20}x_{t,i}-\sum_{i=1}^{20}x_{1,i})+(x_{t,20}-x_{1,20})+(n_t-n_1)+ \\ K(x_{t,20})-K(x_{1,20})+\max(temp\_Tnum_{t,j})-\max(temp\_Tnum_{1,j}) \tag{3-14}$$

现场指纹多于16个时，细节点匹配模块以最慢的速度,不间断地处理前(t-1)枚指纹的全部基偏移量，花费的时间是

$$12\sum_{i=1}^{t-1}d_i+400(t-1)+4\sum_{i=1}^{t-1}\min(d_i,80) \tag{3-15}$$

$(3-14)-(3-15)$，得

$$d_t(d_t+2)+\sum_{i=2}^{t-1}d_i(d_i-11)-13d_1-388(t-1)+2(\sum_{i=1}^{20}x_{t,i}-\sum_{i=1}^{20}x_{1,i})+(x_{t,20}-x_{1,20}) \\ +(n_t-n_1)+K(x_{t,20})-K(x_{1,20})+\max(temp\_Tnum_{t,j})-\max(temp\_Tnum_{1,j})- \\ 4\sum_{i=1}^{t-1}\min(d_i,80),t\geq 3 \tag{3-16}$$

任取 t，在$127 \geq d > 40$，(3-16) 是$d_i, 2 \leq i \leq t$的增函数，d_1的减函数，取

$d_2 = d_3 = \cdots\cdots = d_t = 40, d_1 = 127$，带入 (3-16)，并取

$$n_1 = 20, x_{1,1} = x_{1,2} = \cdots\cdots = x_{1,20} = 12, n_t = 1, x_{t,20} = 0, \sum_{i=1}^{20} x_{t,i} = 1, \max(temp\_Tnum_{t,j}) = 1,$$

$$\max(temp\_Tnum_{1,j}) = 64$$

有

$$f(t) = 612t - 2474, t \geq 3;$$

类似的，由 (3-13) 和 (3-15)，得

$$f(2) = -1260 \qquad (3\text{-}17)$$

$f(t)$的直观意义是细节点匹配之前模块的处理速度与细节点匹配模块的处理速度之差。$f(t)$是关于 t 的增函数，说明随着 t 的增加，细节点匹配的速度会快于前级的速度，流水线能够正常工作，但$f(t)$在$t \leq 4$时小于 0，说明第 4 枚库指纹的第一个有效基偏移量到达细节点匹配模块的时刻早于细节点匹配模块完成对前 3 枚库指纹的有效基偏移量的处理，因此要拿一个 FIFO 来存放没来得及处理的数据。$f(t)$在 t=2 取最小值，说明只要保证 FIFO 的深度足以存放第二枚库指纹的全部有效基偏移量，以后库指纹的数据就不会丢失。将

$$t = 2, d_2 = 40, d_1 = 127, n_1 = 20, x_{1,1} = x_{1,2} = \cdots\cdots = x_{1,20} = 12, n_2 = 1, x_{2,20} = 0, \sum_{i=1}^{n_2} x_{2i} = 1$$

带入 (3-13)，得 982，从第一枚库指纹输出第 1 个有效基偏移量到第 2 枚库指纹输出第 4 个有效基偏移量，经过 982+3=985 个时钟。细节点匹配模块处理第 1 枚库指纹的每个有效基偏移量要花费 560+1=561 个时钟，因此在第一枚库指纹的第一个基偏移量有效起的 985 个时钟内，首先写入 FIFO4 个有效基偏移量，再读出 2 个，最后又写入 4 个，FIFO 的深度最小是 4-2+4=6。

现场指纹特征点少于 17 个时，细节点匹配模块处理前 (t-1) 枚指纹的全部基偏移量的最长时间是

$$4\sum_{i=1}^{t-1} d_i + 1408(t-1) \qquad (3\text{-}18)$$

(3-14) - (3-18)，得

$$d_t(d_t+2)+\sum_{i=2}^{t-1}d_i(d_i-3)-1406(t-1)-5d_1+2(\sum_{i=1}^{20}x_{t,i}-\sum_{i=1}^{20}x_{1,i})+(x_{t,20}-x_{1,20})$$
$$+(n_t-n_1)+K(x_{t,20})-K(x_{1,20})+\max(temp\_Tnum_{t,j})$$
$$-\max(temp\_Tnum_{1,j}),t\geq 3 \qquad (3\text{-}19)$$

任取 t，在$127\geq d\geq 40$，（3-16）是$d_i,2\leq i\leq t$的增函数，d_1的减函数，取$d_2=d_3=\cdots\cdots=d_t=40,d_1=127$，带入（3-16），并取

$$n_1=20,x_{1,1}=x_{1,2}=\cdots\cdots=x_{1,20}=12,n_t=1,x_{t,20}=0,\sum_{i=1}^{20}x_{t,i}=1,\max(temp\_Tnum_{t,j})=1,$$
$$\max(temp\_Tnum_{1,j})=64$$

有

$$f(t)=74t-1090,t\geq 3$$

类似的，由（3-13）和（3-18）得，$f(2)=-934$。

同现场指纹特征点数多于 16 个时的分析类似，从第一枚库指纹输出第 1 个有效基偏移量到第二枚库指纹输出第 4 个有效基偏移量，经过 985 个时钟。细节点匹配模块处理每个偏移量的时间是 478 个时钟，因此从第一枚库指纹的第一个基偏移量有效起的 985 个时钟内，先写入 FIFO4 个有效基偏移量，再读出 3 个，最后再写入 4 个，FIFO 的深度最小是 4-3+4=5。

兼顾现场指纹特征点多于 16 个和少于 17 个两种可能，FIFO 深度最小是 6。

现场指纹特征点多于 32 个，库指纹的有效基偏移量有 4 个时，从边产生模块发给边计算模块的指纹切换信号 re_comp_1_2 有效，到细节点匹配模块完成对该库指纹特征点的最后一次访问，经过了

$$(3-12)+3\times(3-5)+3+41+(d+2)\lceil f/32\rceil$$

(3−12)是公式（3-12），(3−5)是公式（3-5），整理得

$$3ptos\_count\_match+666+4(d+2)\lceil f/32\rceil$$
$$+2\sum_{i=1}^{20}x_i+x_{20}+n+K(x_{20})+\max(temp\_Tnum_j)+d \qquad (3\text{-}20)$$

取$ptos_count_match=80,d=127,x_i=12,n=20,\max(temp_Tnum)=64$，得到（3-20）的最大值 3157，而（3-1）的最小值是 1642，说明在细节点匹配模块读库指纹的过程中，库指纹的读指针有可能被更新，读的不再是想要的库指纹。

本文提出的办法是，在库指纹的每个读指针被更新前，也就是在它被更新后的 1642 个时钟周期内，保存可能会用到的读指针。如果库指纹没有生成有效偏移量，那么后续的细节点匹配模块就不会重新读取该库指纹，保存该库指纹的地

址就没有意义，所以应在偏移量生成模块输出第一个有效基偏移量之后保存读指针。

选择在细节点匹配模块从FIFO中读出第一个偏移量的时候保存当前库指纹在多重乒乓 RAM 的地址。从边产生模块发给边计算模块的指纹切换信号 re_comp_1_2 有效，到细节点匹配模块保存当前库指纹在多重乒乓RAM的地址的时间间隔是(3−12)+4，即

$$2\sum_{i=1}^{20}x_i + x_{20} + 347 + n + K(x_{20}) + \max(temp\_Tnum_j) + d \qquad (3\text{-}21)$$

同以上分析类似，（3-21）的最大值是1038，此时距多重乒乓RAM上次被更新有1038-1=1037个时钟，因为它小于多重乒乓RAM指针更新周期的最小值1642，所以这段时间内读指针没有被更新。

如果现场指纹特征点数目少于17个，和现场指纹特征点数目多于16个只在最后的打分筛选算法上有根本区别。细节点匹配模块中前面各级模块的延迟随现场指纹特征点的减少而缩短，所以现场指纹特征点少于17个时，以上分析依然成立。

整个系统工作于流水线方式，当特征三角形比对模块边生成模块后的部分在处理一枚库指纹时，偏移量计算模块在处理上一枚库指纹，细节点匹配模块在处理上上一枚，所以采用3重乒乓RAM暂存库指纹就够用了。

系统复位后，写指针和读指针的值分别是“00”和“10”，以后每当re_comp有效，写指针更新为旧值加1再模3，读指针更新为写指针的旧值。读指针rd_ptr输出给现场指纹旋转平移模块，在偏移量计算模块输出的偏移量中，增加标志位指示是否是一枚库指纹的第一个偏移量。偏移量读取控制模块读出一个偏移量后检查它是否是第一个，如果是第一个，置offset_first为‘1’，现场指纹旋转平移模块检查到offset_first是‘1’，就把rd_ptr保存到rd_ptr_lock里，后面的库指纹读取控制模块使用rd_ptr_lock，而非rd_ptr，作为读取库指纹的指针。

3.6 对偏移量读取控制的改进

3.6.1 原设计的偏移量读取模块的问题

图3-3中的偏移量读取控制模块从FIFO读出有效基偏移量。

原设计的偏移量读取控制模块受三个信号控制：偏移量存储FIFO空标志信号empty，对应于前一个偏移量的库指纹读取结束标志信号dfb_over，打分筛选模块发送的允许信号offset_read_valid。原设计的状态机在复位之后进入idle，在

idle 状态检测 empty，当 empty='0'时进入 st_read 并从 FIFO 读出偏移量，随后进入等待状态 wt_over：当 dfb_over 和 offset_read_valid 都有效后，回到 idle。在处于 st_read 和 wt_over 时，检测到 empty='1'则直接回到 idle。这样做存在两个问题：

（1）dfb_over 是后级模块的启动信号，其中打分筛选模块也间接受其控制，也就是说，dfb_over 总是先于 offset_read_valid 有效，使得原设计中对 dfb_over 电平的判断过程没有意义。

（2）在状态机处于 st_read 和 wt_over 时，检测到 empty='1'，即 FIFO 是空的，就直接回到 idle，但此时 offset_read_valid 不一定有效。状态机回到 idle 后，如果偏移量生成模块向 FIFO 写入新的有效基偏移量，那么又会启动新一轮的细节点匹配，如果上一个有效基偏移量的 offset_read_valid 还未有效——对上一个有效基偏移量的计算尚未结束，就会造成工作混乱。

3.6.2 新的偏移量读取控制

新的偏移量读取控制模块的状态机如图 3-18 所示：

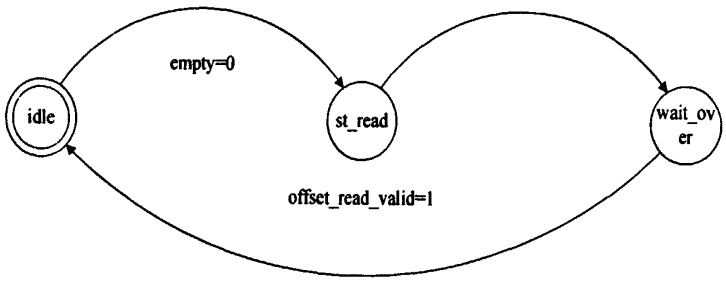

图 3-18 偏移量读取控制状态机

复位信号 rst 有效后，状态机进入 idle，探测到 empty='0'就进入 st_read，在 st_read 发有效的 FIFO 读信号 offset_rdreq，一个时钟后状态机从 st_read 无条件进入 wait_over，在 wait_over 只要探测到 offset_read_valid='1'就返回 idle。

offset_rdreq 寄存后作为 FIFO 输出有效基偏移量 offset_fifo_data 有效的信号，offset_fifo_data(50)指示该有效基偏移量是否是一枚库指纹的第一个有效基偏移量。

3.6.3 新的偏移量读取控制与原设计的比较

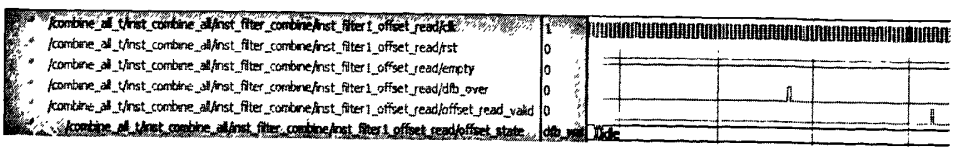

图 3-19 原始偏移量读取控制的功能仿真

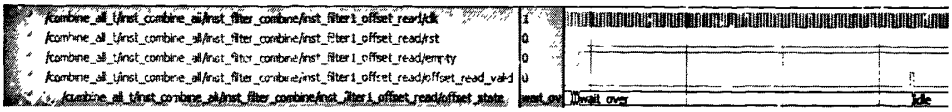

图 3-20 新的偏移量读取控制的功能仿真

比较图 3-19 和图 3-20 可以发现，原始的偏移量读取控制状态机在 empty='1' 后进入 idle，而此时后级尚未返回有效的 dfb_over 和 offset_read_valid，如果 FIFO 在 offset_read_valid 有效前又有新的有效基偏移量要求处理，状态机工作就会进入混乱。而新的设计就避免了这个问题，empty='1'后直到 offset_read_valid 有效后才回到 idle，使得对下一个有效基偏移量的计算在上一个完成之后开始。

取默认设置，在 QuartusII 9.1 环境下的编译报告如图 3-21 至 3-22 所示：

Flow Status	Successful - Fri Feb 05 13:53:42 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	filter_offset_read
Top-level Entity Name	filter_offset_read
Family	Stratix II
Device	EP2S90F1020C5
Timing Models	Final
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	7 / 72,768 (< 1 %)
Dedicated logic registers	65 / 72,768 (< 1 %)
Total registers	65
Total pins	115 / 759 (15 %)
Total virtual pins	0
Total block memory bits	0 / 4,520,448 (0 %)
DSP block 9-bit elements	0 / 384 (0 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

图 3-21 原始偏移量读取控制的资源占用

	Slack	Actual fmax (period)	From	To	From Clock	To Clock	Required Setup Relationship	Required Longest P2P Time	Actual Longest P2P Time
1	N/A	230.26 MHz (period = 4.343 ns)	flag_read	offset_theta[5]~reg0	clk	clk	None	None	4.113 ns
2	N/A	231.48 MHz (period = 4.320 ns)	flag_read	offset_theta[7]~reg0	clk	clk	None	None	4.058 ns
3	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[14]~reg0	clk	clk	None	None	3.646 ns
4	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[15]~reg0	clk	clk	None	None	3.646 ns
5	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[16]~reg0	clk	clk	None	None	3.646 ns
6	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[17]~reg0	clk	clk	None	None	3.646 ns
7	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[18]~reg0	clk	clk	None	None	3.646 ns
8	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[19]~reg0	clk	clk	None	None	3.646 ns
9	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[20]~reg0	clk	clk	None	None	3.646 ns
10	N/A	258.00 MHz (period = 3.876 ns)	flag_read	offset_out[21]~reg0	clk	clk	None	None	3.646 ns

图 3-22 原始偏移量读取控制的前 10 条最大延时路径

Flow Status	Successful - Fri Feb 05 13:46:54 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	filter1_offset_read
Top-level Entity Name	filter1_offset_read
Family	Stratix II
Device	EP2S90F1020C5
Timing Models	Final
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	3 / 72,768 (< 1 %)
Dedicated logic registers	64 / 72,768 (< 1 %)
Total registers	64
Total pins	116 / 759 (15 %)
Total virtual pins	0
Total block memory bits	0 / 4,520,448 (0 %)
DSP block 9-bit elements	0 / 384 (0 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

图 3-23 新的偏移量读取控制的资源占用

	Slack	Actual fmax (period)	From	To	From Clock	To Clock	Required Setup Relationship	Required Longest P2P Time	Actual Longest P2P Time
1	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[15]~reg0	clk	clk	None	None	3.322 ns
2	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[16]~reg0	clk	clk	None	None	3.322 ns
3	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[17]~reg0	clk	clk	None	None	3.322 ns
4	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[18]~reg0	clk	clk	None	None	3.322 ns
5	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[19]~reg0	clk	clk	None	None	3.322 ns
6	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[20]~reg0	clk	clk	None	None	3.322 ns
7	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[21]~reg0	clk	clk	None	None	3.322 ns
8	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[22]~reg0	clk	clk	None	None	3.322 ns
9	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[23]~reg0	clk	clk	None	None	3.322 ns
10	N/A	276.40 MHz (period = 3.618 ns)	offset_rdreq_dly	offset_out[24]~reg0	clk	clk	None	None	3.322 ns

图 3-24 原始偏移量读取控制的前 10 条最大延时路径

通过对图 3-21 至图 3-24 的观察可见，新的偏移量读取控制比原设计占用的资源少了，最高工作频率提高了。这是通过分析工作时序对原始设计状态机进行优化（从 5 个状态减少到 3 个状态），并采取适合 FPGA 的代码风格（用 registers 代替 ALUTs）实现的。

3.7 对库指纹特征点读取控制的修改

因为引入了库指纹序号 FID 和重写了三重乒乓 RAM，所以对图 3-3 的库指纹特征点读取模块做了两处修改：

（1） 在读取库指纹特征点的地址信号中加上了三重乒乓 RAM 的寻址信号 rd_ptr_lock， rd_ptr_lock 的意义见 3.5.3 节。

（2） 增加了提取 FID 的功能。

由于三重乒乓 RAM 中最多可同时存放 3 枚库指纹的数据，所以传给三重乒乓 RAM 的地址不但要包含一枚库指纹内的地址，还要指明是哪一枚库指纹，所

以传给三重乒乓RAM的地址filter_dfb_rdaddress由rd_ptr_lock&count形成，count用来区分一枚库指纹内部的地址。

如图 3-25 所示，状态机在 read_max 给出当前库指纹的首地址（count="0000000"），存储空间的首地址的高 25 位记录该库指纹的序号 FID，两个时钟后把 FID 提取出来并连到储存比对结果的 FIFO 的控制器。其余与原设计相同。

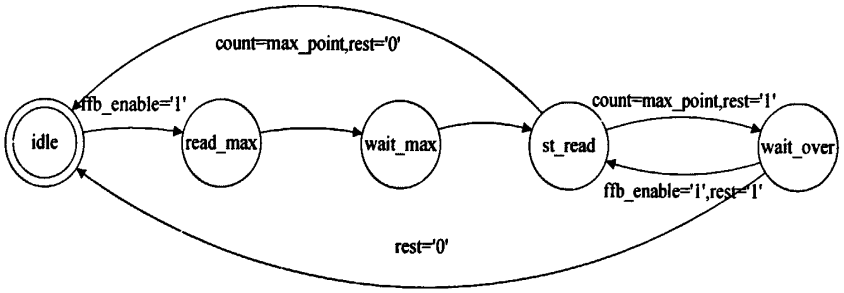

图 3-25 库指纹读取控制模块状态转移图

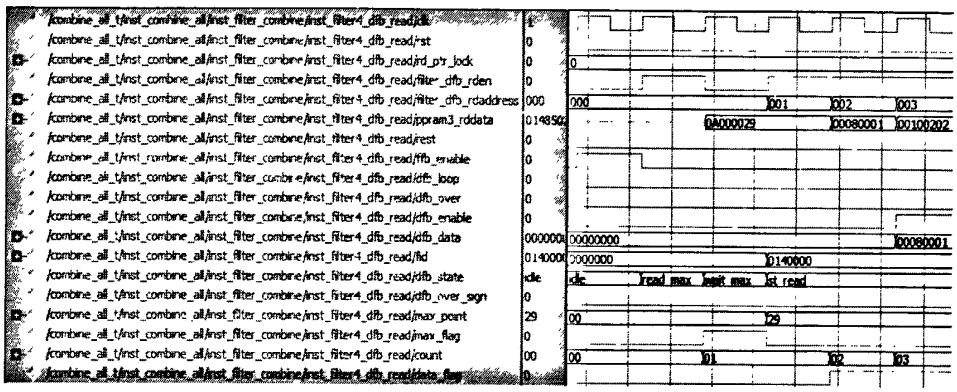

图 3-26 库指纹读取控制模块功能仿真

从图 3-26 可见，在 read_max 读第一枚库指纹的第一个点（rd_ptr_lock=2b0，filter_dfb_rden=1b1,filter_dfb_rdaddress=9b0），其中存放的库指纹序号和特征点数两个时钟后分别存入 FID 和 max_point 里。

3.8 比对结束信号生成模块 matchover_gen

3.8.1 为什么用全'1'的 FID 指示比对结束

原设计中，FPGA 内部的核心比对电路与应用程序之间没有信号线指示比对结束。因此需要用其他办法结束比对。

受本课题 PCB 板硬连线的限制，FPGA 芯片不能通过写 9054 的 mailbox 寄存器或 doorbell 寄存器设置比对结束的信号。因此要用其他办法实现相同功能。

9054 与 FPGA 之间的数据交互有现场指纹、库指纹、查找表、相关参量及匹配库指纹 ID。在比对过程中，只有库指纹数据和匹配库指纹 ID 是可更新的，因此可以在库指纹和匹配库指纹 ID 的数据结构中设置标识位来结束比对。本文提出的办法是用库指纹序号 FID 传递比对完成信号，FID 是 25 位二进制数，FID 是全'1'时，表示当前库指纹是最后一个，ID 为全'1'的库指纹不是真实的，它存在的唯一意义是传递比对结束的信号。

如果核心比对电路检测到全'1'的 FID，就在完成比对最后一枚合法库指纹的时候结束比对。由于 FID 为全'1'的库指纹未必与现场指纹匹配，所以需要单独设计一个独立于核心比对模块的电路 matchover_gen 用来生成比对结束信号 matchover。

3.9.2 matchover_gen 的设计

核心比对电路收到全'1'的库指纹时，向控制存放比对结果的 FIFO 的模块发结束比对信号 matchover，控制存放比对结果的 FIFO 的模块检测到 matchover='1'时，将全'1'的 FID 写入 FIFO，主机端的应用程序检测到 FIFO 返回全'1'的 FID 之后完成比对。

将全'1'的 FID 转化为内部比对结束信号 matchover 的电路 matchover_gen 的状态机如图 3-27 所示：

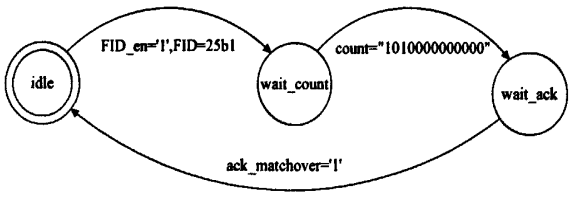

图 3-27 产生比对结束信号 matchover 的模块 matchover_gen 的状态机

系统复位后，状态机进入 idle 状态，在 idle 接收边生成模块传来的库指纹序号 FID 及其使能信号 FID_en，FID_en='1'且 FID=25b1 说明比对已经结束，进入 wait_count。

整个系统工作于流水线模式，因此在 matchover_gen 接收到比对结束信号后，还要等到当前核心比对模块中的合法库指纹处理完毕，再发 matchover 给写 FIFO 控制模块 fifo_write。核心比对模块要求库指纹特征点不得少于 40 个，在此约束下边产生模块遍历 1 枚库指纹最少需要 1642 个时钟，而特征三角形比对需要 21+40=61 个时钟，偏移量计算和细节点匹配花费的时间都少于边产生模块的周期，在细节点匹配完成后（offset_read_valid='1'），最多经过 18 个时钟后 match

有效，所以状态机在收到全'1'的 FID 后，经过1642×3+61+18=5005个时钟，所有合法库指纹必定处理完毕。为计数方便，在 wait_count 等待 4096+1024+1=5131 个时钟。状态机进入 wait_count 时，count=13b0，每过一个时钟 count 加 1，在某一时钟沿，检测到 count="1010000000000"时退出 wait_count，进入 wait_ack。

在 wait_ack 置 matchover 为'1',等待 fifo_write 发送 ack_matchover，ack_matchover='1'表示 fifo_write 受到终止比对的请求，状态机返回 idle。

图 3-28 是 matchover_gen 的功能仿真波形：

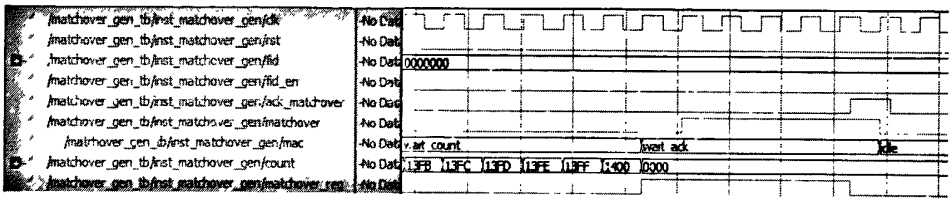

图 3-28 matchover_gen 的功能仿真波形

关于比对结束信号的生成，还要做一些补充。

9054 向 FPGA 每次传 2000 枚库指纹，FPGA 再传到片外 SRAM。如果指纹库内指纹数量不是 2000 的倍数，最后一批库指纹就不够 2000 枚，片外 SRAM 控制器会认为当前这一批库指纹传输尚未结束，结果一直等待下去，dfb_ready 也保持为'0'，核心比对模块也停止工作。

因此，如果最后一批库指纹不够 2000 枚，就要在保证不影响匹配结果的前提下凑成 2000 枚。 FIFO 写控制模块 fifo_write（见下节）自动忽略重复的匹配库指纹。因此可采用以下方法处理最后一批库指纹，使之成为 2000 枚。

（1） 如果库指纹总数为 2000n+1999，那么应用程序在最后一枚合法库指纹后面再加上一枚 ID 是全'1'的库指纹。

（2） 如果库指纹总数为 2000n+m，$0 \le m < 1999$，那么应用程序在最后一枚合法库指纹后面再添加它的 1999-m 个副本，末尾再加上一枚 ID 是全'1'的库指纹。

3.9 存放匹配库指纹序号的 FIFO 的写控制模块 fifo_write

3.9.1 fifo_write 的作用

核心比对模块输出比对结果——匹配信号 match 和库指纹序号 FID，返回给主机。由于从输出比对结果给 9054，到 9054 读取比对结果，这中间的延时是不确定的，有可能很大，所以采用一个 FIFO 作为缓冲，存放匹配的 FID。采用 FIFO 的另一个原因是比对过程中的库指纹和比对结果都是通过 9054 与 FPGA 间的本

地总线传输的，9054 每次请求占用本地总线的过程都需要一些握手信号完成，如果每产生一个匹配结果就要求 9054 来读取，势必造成 9054 对本地总线的频繁访问，导致传输效率低下。利用 FIFO 暂存匹配库指纹 FID，待到 FIFO 充满到一定程度再发中断给 9054，请求 9054 一次读取多个匹配 FID 就可以避免上述弊端。

3.9.2 fifo_write 的设计

原来的设想是通过片外 FIFO 存放匹配库指纹 FID，FPGA 仅仅产生对它的控制信号。但在开发板的设计方案中，片外 FIFO 的 LD#被接地了，导致它始终处于等待配置状态，无法利用里面的存储资源。所以改用片内 FIFO 来完成暂存匹配结果的功能。

fifo_write 等到有效的 match 输出后，决定是否将 FID 写入 FIFO 内，而细节点匹配模块每次启动新的偏移量读取是由 offset_read_valid 触发的，offset_read_valid 提前于 match 有效而有效，从 offset_read_valid 有效到 match 有效这段时间内，细节点匹配模块内的库指纹读取模块有可能将 FID 更新，所以在 fifo_write 内部，在 offset_read_valid='1'时将 FID 存入 FID_reg，以后往 FIFO 里写的是 FID_reg 而不是 FID。如图 3-29 所示：

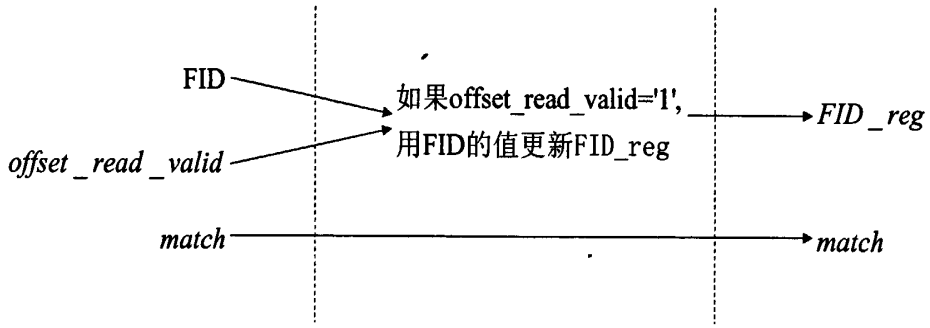

图 3-29 寄存 FID 的数据流图

fifo_write 的状态转移如图 3-30 所示：

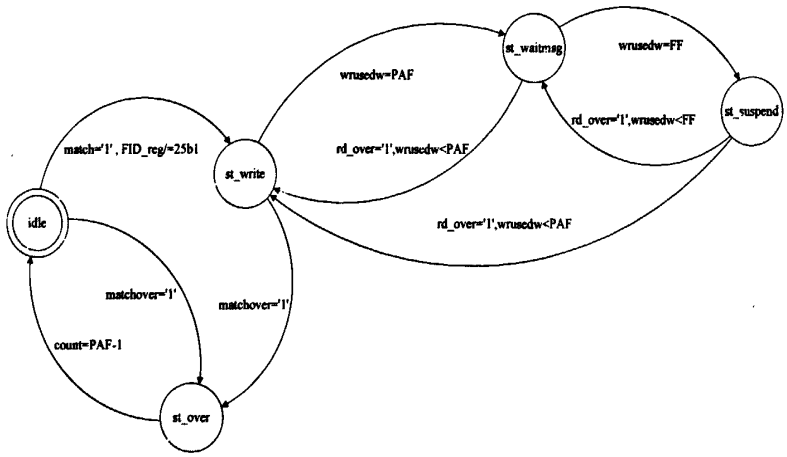

图 3-30 fifo_write 的状态转移图

系统复位信号 rst='1'时，状态机进入 idle 状态，在 idle 状态检测到 match='1'，并且 FID_reg 不等于 25b1，说明有一枚合法的库指纹与现场指纹匹配，fifo_write 此刻将 FID_reg 写入 FIFO 里，同时用 FID_reg 更新内部寄存器 oldFID，随后状态机进入 st_write 状态。

match 的含义是现场指纹以当前有效基偏移量变换到库指纹坐标系后，与库指纹是否匹配，每一枚库指纹最多可生成 4 个有效基偏移量，也就是说每一枚库指纹可生成多个有效的 match。如果只依 match 将 FID 写入 FIFO，就可能将一枚库指纹 FID 重复写入 FIFO 里，oldFID 的存在就是为了解决这个问题：除了写第一枚库指纹 FID，其他时刻都要满足 FID_reg 不等于 oldFID 才把 FID_reg 写入 FIFO，同时用 FID_reg 更新 oldFID。

在 st_write 状态，状态机跟据 FID_reg、match 及 oldFID 控制对 FIFO 的写操作，wrusedw 是 FIFO 传过来的信号，指示 FIFO 里已被占用了多少个存储单元。状态机检测到 wrusedw=PAF(Programmable Almost Full)时，说明 FIFO 已充满到一定程度，发中断信号 intarrupt 给 9054 后进入 st_waitmsg 状态。

为提高比对速度，fifo_write 不是等到 FIFO 充满，而是到 wrusedw=PAF 时就给 9054 发中断，这样在 9054 读 FIFO 时 fifo_write 仍然可以向 FIFO 里写数据。但从 fifo_write 给 9054 发中断，9054 再把中断交给主机处理，一直到主机响应中断，这段延时是不确定的，主机如果忙于其他任务，延时可能会很长，在 st_waitmsg 核心比对电路仍在工作，因此 FIFO 仍有被填满的可能。为防止写入 FIFO 的数据溢出，在恰当时刻应停止核心比对电路工作。系统工作在流水线模式，FIFO 被充满时，核心比对电路可能还在处理其他库指纹，因此要在 FIFO 被充满前留出一定提前量。边产生模块在遍历一枚库指纹时，特征三角形比对、偏移量计算和细节点匹配模块依次在处理该库指纹之前的第一、第二和第三枚库指

纹，因此要在 FIFO 还剩下 4 个可用存储单元时停止核心比对电路工作。故在 st_waitmsg 时要检查 wrusedw 的值，在 wrusedw=FF(FF=FIFO 深度-4)时发 suspend 给边生成模块，同时自己转移到 st_suspend。边生成模块如何处理 suspend 见 3.1 节。

反之如果主机端及时响应，那么 FIFO 就没有被充满的可能。为提高本地总线利用率，9054 采用 DMA 方式读取匹配结果，FPGA 检测到 9054 完成一次 DMA 后，会置 rd_over='1'。状态机在 st_waitmsg 检测到 rd_over='1'，并且 wrusedw<PAF 就返回到 st_write。

状态机在 st_suspend 保持 suspend 有效，继续接收最多 4 枚匹配库指纹，同时等待 rd_over='1'，由于读和写 FIFO 的速率变化都较快，因此 rd_over='1'时如果 wrusedw<PAF，状态机回到 st_write；如果 PAF<wrusedw<FF，回到 st_waitmsg，同时再发一个中断；如果 $wrusedw \geq FF$，就留在 st_suspend，同时也要再发一个中断给 9054。

在 st_write，如果 matchover='1'，而且 wrusedw<PAF，就转移到 st_over。由于 9054 用 DMA 方式每次读取 PAF 枚库指纹，所以 st_over 是专门为了把最后不够 PAF 个库指纹用非法 FID 凑成 PAF 个而设的状态。在 st_write 向 st_over 转移的同时，还要回复 ack_matchover 并把 wrusedw 的值存到 count 里，在 st_over 每个时钟向 FIFO 写入一个全'1'的 FID，同时把 count 加 1，检测到 count=PAF-1 说明 FIFO 里共有 PAF 枚库指纹（包括非法的），回到 idle。

即使库里没有一枚匹配的库指纹，也要在 matchover='1'时由 idle 进入 st_over，因为核心比对电路不把非法 FID 返回主机的话，主机是不会知道是因为没有匹配库指纹还是核心比对电路尚未完成计算而导致迟迟不响应的。状态机由 idle 进入 st_over 的一系列操作与由 st_write 进入 st_over 相同。

fifo_write 的功能仿真波形如图 3-31 所示：

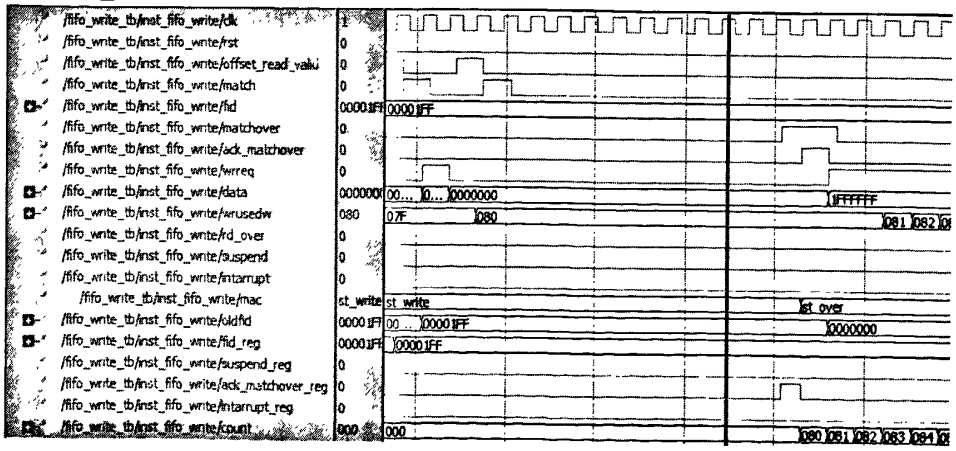

图 3-31 fifo_write 的功能仿真波形

3.10 对打分模块 filter_marker 的修改

3.10.1 打分筛选模块的作用

图 3-3 中，打分筛选模块完成粗筛选，是细节点匹配中最重要的一个模块，也是运算最复杂的一个模块。该模块按照现场指纹细节点的数目 mNum 被分成两部分，mNum≥17 和 mNum<17，之所以这样来划分是因为当现场指纹细节点数目比较多时（mNum≥17），大致按照匹配点对的数目来打分筛选就可以满足筛选的需要，而且这样做在硬件实现上可以节省运算时间，保证流水线不"溢出"；而当现场指纹细节点数目较少时（mNum<17），匹配点对的数目能提供的信息太少，达不到粗筛选的目的，此时需要统计匹配点对之间的相对信息，利用匹配点对及其相对信息完成粗筛选是比较合理的方法，而且由于现场指纹细节点数目较少，硬件实现时也不会因为其运算复杂、时间过长而导致流水线溢出。

原设计中根据 mNum 的值有效 mark_up_valid 或 mark_below_valid，选通 mark_up 模块或 mark_below 模块，分别实现对现场指纹细节点多于 16 个及少于 17 个的打分筛选。

3.10.2 原设计中的 filter_marker 的问题及解决

原设计取现场指纹细节点数目 ffb_num 的低 5 位作为 mNum 与 17 比较，小于 17 时，选通少于 17 个的打分模块 filter_marker_below，反之选通多于 16 个的打分模块 filter_marker_up。取 ffb_num 的低 5 位相当于求 ffb_num 除以 32 的余数，从 ffb_num 除以 32 的余数与 17 的关系不能必然得出 ffb_num 与 17 的关系。（例如，ffb_num=40，mNum=8）。

因此做如下修改：在 filter_marker 里用 ffb_num 直接与 16 比较，大于 16 就选通 filter_marker_up，反之选通 filter_marker_below。filter_marker_below 里面用到的库指纹特征点数目取 ffb_num 的低 5 位，由于这时 ffb_num<17，所以这样做不会发生错误。

在 modelsim SE PLUS 6.5 环境下，取 ffb_num=40，对原设计和修改后的设计的功能仿真分别是图 3-32 和图 3-33。图 3-32 输出报告选通 filter_marker_below，图 3-33 输出报告选通 filter_marker_up。

```
# ** Note: filter_marker_below being used
#    Time: 69889600 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_below being used
#    Time: 69903752 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_below being used
#    Time: 69918904 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_below being used
#    Time: 69934056 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_below being used
#    Time: 69949208 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_below being used
#    Time: 69964360 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
```

图 3-32 原设计的 filter_marker 的功能仿真报告

```
# ** Note: filter_marker_up being used
#    Time: 69888620 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_up being used
#    Time: 69913752 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_up being used
#    Time: 69918974 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_up being used
#    Time: 69934056 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_up being used
#    Time: 69949218 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_up being used
#    Time: 69964360 ps  Iteration: 1  Instance: /combine_all_t/inst_combine_all/inst_filter_combine/inst_filter9_marker
# ** Note: filter_marker_up being used
```

图 3-33 修改后的 filter_marker 的功能仿真报告

3.11 对现场指纹特征点少于 17 个的求结构分模块 filter_marker_total_grade 的改进

3.11.1 filter_marker_total_grade 的作用

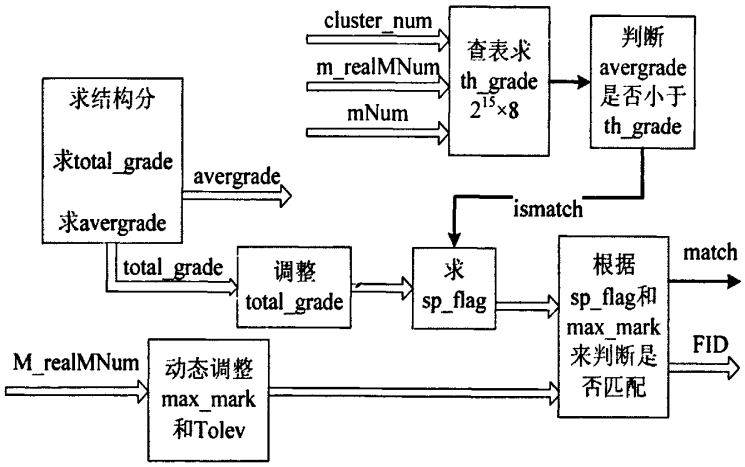

图 3-34 mark_below 子模块内部结构示意图【5】

filter_marker_total_grade 位于图 3-3 的打分筛选模块的 mark_below 子模块的内部，当现场指纹细节点个数少于 17 个时被选通。

图 3-34 中，第一步求取结构分是打分筛选算法中的一个关键步骤，结构分表征各个匹配点对之间的相对关系，是打分筛选需要考虑的一个重要因素。

一个匹配点对实际对应着现场指纹和库指纹细节点各一个，两个匹配点对便可以构成两条边，一条现场指纹边和一条库指纹边，考虑这两个匹配点对的相对信息，如果这两个匹配点对构成的两条边的匹配分值比较高，那么这两个匹配点对的可信度也会相应增加。求取结构分正是利用这样的思路，针对每一个匹配点对，在其周围（现场指纹边的长度不超过 120）最多找三个匹配点对（有可能小于 3），然后对这三对边分别进行打分，根据三个分值之和按照一定的规则就可以求得该匹配点对的结构分。求取结构分采用了流水线的操作方法，其实现流程

的示意图如图 3-35 所示，其中结构分打分采用了两级查表的结构，这是因为打分涉及到了变量的除法运算。

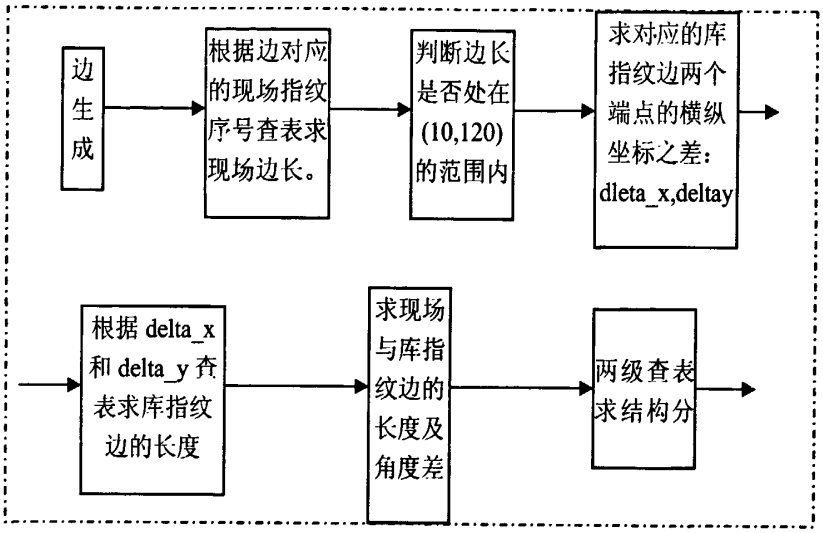

图 3-35 结构分求取流程示意图[5]

3.11.2 原设计的 fiter_marker_total_grade 的问题

原设计中为减少电路面积，在进行二重循环求结构分前将 m_realMNum 减 1 得到 match_num，同时将聚类点统计模块发来的启动比对信号 match_below_valid 延时一个时钟与 match_num 同步。状态机在 idle 接收到延时一个周期的 match_below_valid 后，判断 m_realMNum 是不是 0，m_realMNum 不是 0 的话，说明经过图 3-14 的匹配点对调整后依然有匹配点对需要处理，进入 st_count1，在 st_count1 状态判断 match_num 与计数器 count1 是否相等，count1 是匹配点对现场指纹特征点外循环点计数器，在 idle 时是 0，如果 count1=match_num（m_realMNum-1)说明外循环点遍历到了最后一个点，count1 接着变成 0，遍历第一个点，如果 count1 不等于 match_num，则首先将 count1 自增 1，进入 st_count2，st_count2 完成内循环点循环。大多数情况下 match_num 不是 0，因此一般来说外循环点首先是从第二个点开始遍历的(count1=1)。但是当 match_num 等于 0 时，就出现问题了。

count1=match_num1=0 时，与库指纹有对应匹配点的现场指纹特征点只有一个。状态机进入 st_count2 开始遍历现场指纹中以该点为一个端点的边，该点也是第一个外循环点(m_realMNum=1)，因此退出 st_count2 的条件是 count2=match_num-1，不满足此条件则每过一个时钟将 count2 加一。match_num 是 4 位二进制有符号数，match_num="0000"时，match_num-1="1111"，翻译成有符号数是-1，而 count2 这时是 0，因此要再经过 15 个时钟后，count2="1111"才返回 idle。状态机处于 st_count2 时用 count2 作为读内循环点的指针，这时有

匹配点对的现场指纹特征点只有 1 个，放在聚类点统计模块的两块乒乓 RAM 中地址为 0 的单元，count2 从“0001”递增到“1111”访问的是无效的数据，因此得到的结构分也是错误的。

3.11.3 新的求结构分状态机

新的求结构分状态机状态转移图如下图：

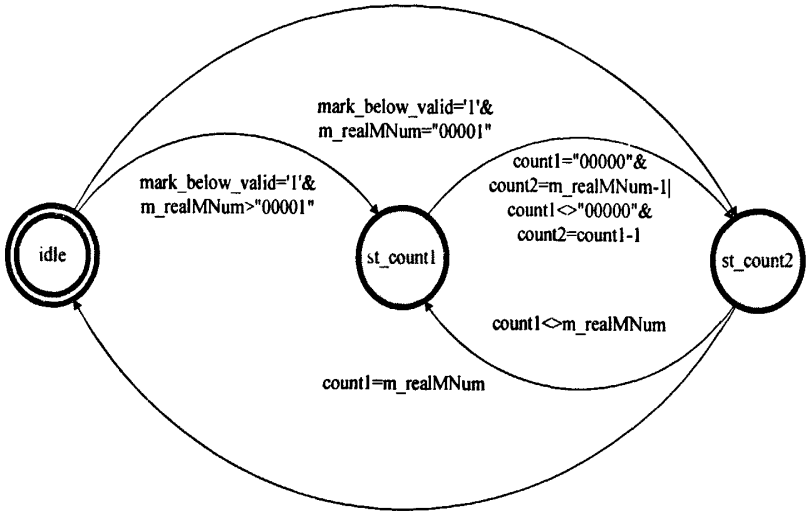

图 3-36 求结构分状态机

如图，状态机的初始状态是 idle，在 idle 状态收到聚类点统计模块发过来的指示打分信号 mark_below_valid 的同时判断 m_realMNum 的值，m_realMNum 是 0，说明前面产生没有匹配点对，保持 idle 不变，并发 offset_read_valid_below 给偏移量读取控制模块，指示读取下一个有效基偏移量；m_realMNum 是 1，说明只有一个匹配点对，不必进行二重循环求结构分，直接跳到 st_count2，产生 offset_read_valid_below；m_realMNum 大于 1，说明可以两两组合成现场指纹及库指纹的边求结构分，置 flag1='1'，指示后面的流水线当前产生的是外循环点，并有效 match_rden 读取第一个外循环点，然后跳到 st_count1 开始以第一对匹配点对的现场指纹特征点为外循环点的循环。

在 st_count1，有效 flag2，状态机用计数器 count2 完成对内循环点的循环，求结构分是内外循环点两两组合形成边的过程，因此要避免内外循环点相同的情况。每次内循环 count2 从(count1+1) mod m_realMNum 按升序遍历到

$$0 \le count1 \le m\_realMNum-1,$$
$$1 \le count1+1 \le m\_realMNum,$$
$$-1 \le count-1 \le m\_realMNum-2$$

(count1-1) mod m_realMNum。

求模在 VHDL 里一般不可综合，因此代码中要针对操作数超过 0~m_realMNum 的情况做出特殊规定。具体地说，$count+1=m\_realMNum$ 时，

(count+1) mod m_realMNum=0，count1-1=-1，即 count1=0 时，(count1-1) mod m_realMNum=m_realMNum-1，count2 遍历到 m_realMNum-1 时，先修改外循环点 count1<=count+1，然后跳到 st_count2。其他情况(count1-1) mod m_realMNum=count1-1，count2 遍历到 count1-1 时修改 count1 并跳到 st_count2。

在 st_count2 有效 flag3，最后一轮循环 count1 的值是 m_realMNum-1，修改后是 m_realMNum, 因此在 st_count2 可通过判断 m_realMNum 确定求结构分的过程是否结束：count1=m_realMNum 的时候返回 idle，并有效 flag_end，flag_end 经过若干个时钟的延时后作为后级模块判断是否匹配的启动信号；否则当 count1=m_realMNum-1 时，(count1+1) mod m_realMNum=0，count1<>m_realMNum-1 时，(count1+1) mod m_realMNum=count1+1，用(count1+1) mod m_realMNum 更新 count2，并有效 flag1 和 match_rden，然后返回到 st_count1。

结构分是通过不同匹配点对的库指纹特征点和现场指纹特征点组合形成库指纹边和现场指纹边，然后将现场指纹映射到库指纹坐标系里求它们之间的角度和长度差，再经两级查表后得到的。匹配点对只有 1 个时不能求结构分，但匹配分依然有效，这就要求代码能对结构分不存在的情况做出正确处理而不出现异常，状态机后面的求结构分和匹配分之和 total_grade 和匹配分均值 m_avergrade 的流水线保留了原设计方案。具体的说，采用 flag1、flag2、flag3 和 flag_end 作为指示信号。flag1 及其寄存后的信号指示当前处理的是与外循环点有关的数据，对应的计算单元收到有效的 flag1 及寄存后的信号统计匹配分；flag2 及其寄存后的信号指示当前处理的数据与内循环点相关，flag2 或寄存后的信号有效时，进行内外循环点组合形成现场指纹边及判断边的长度是否在 10 和 120 之间；flag3 及其寄存后的信号表示外循环点发生切换，流水线收到有效的 flag3 或寄存后的信号进行结构分的累加；flag_end 表示外循环点遍历结束，流水线这时可以统计最终的 total_grade 和 m_avergrade，前文提到当 m_realMNum=1 直接从 idle 跳到 st_count2 而不经过 st_count1，因此 flag1 首先有效，过 1 个时钟后 flag3 和 flag_end 有效，flag2 一直无效，求结构分的模块就处于“休眠”状态，求 total_grade 时代表结构分的部分为 0。

3.11.4 新的打分模块与原设计的比较

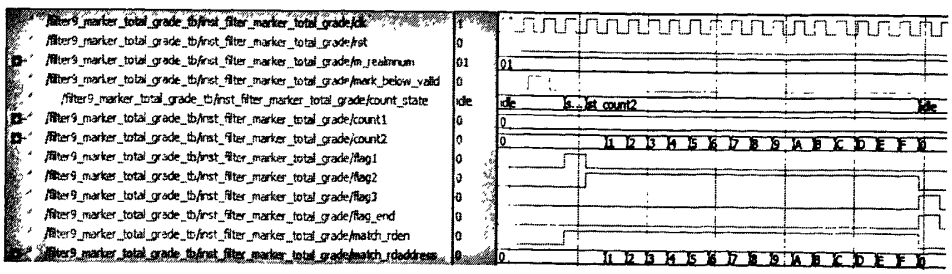

图 3-37 原始的少于 17 个特征点的打分模块的功能仿真

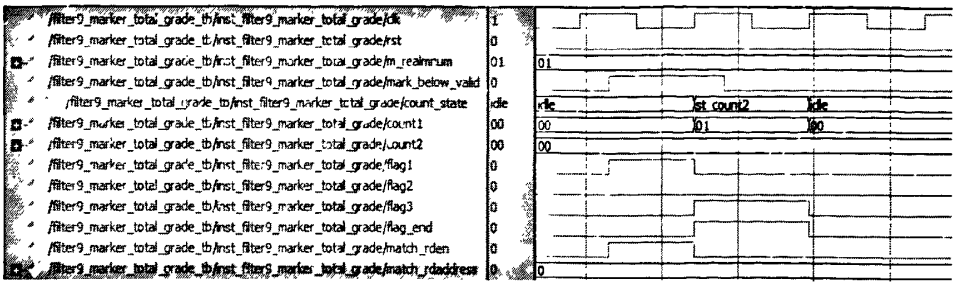

图 3-38 新的少于 17 个特征点的打分模块的功能仿真

从图 3-37 可以看见，当 m_realMNum=1 时，原始设计输出从 0 到 15 的 match_rdaddress，导致读入无效的数据，影响结构分的计算，而 3-38 中新的设计就消除了这个问题。

取 QuartusII 9.1 的默认设置，得到的编译报告如图 3-39~图 3-42 所示。

Flow Status	Successful - Thu Feb 04 17:37:45 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	filter_marker_total_grade
Top-level Entity Name	filter_marker_total_grade
Family	Stratix II
Device	EP2S90F1020C5
Timing Models	Final
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	474 / 72,768 (< 1 %)
Dedicated logic registers	400 / 72,768 (< 1 %)
Total registers	400
Total pins	145 / 759 (19 %)
Total virtual pins	0
Total block memory bits	527,020 / 4,520,448 (12 %)
DSP block 9-bit elements	2 / 384 (< 1 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

图 3-39 原始的少于 17 个特征点的打分模块资源占用率

	Slack	Actual fmax (period)	From	To	From Clock	To Clock	Required Setup Relationship	Required Longest P2P Time	Actual Longest P2P Time
1	N/A	124.49 MHz (period = 8.033 ns)	dfb_theta[2]	temp_marks_rdaddress2[1]	clk	clk	None	None	7.745 ns
2	N/A	124.49 MHz (period = 8.033 ns)	dfb_theta[2]	temp_marks_rdaddress2[2]	clk	clk	None	None	7.745 ns
3	N/A	124.50 MHz (period = 8.032 ns)	dfb_theta[2]	temp_marks_rdaddress2[3]	clk	clk	None	None	7.744 ns
4	N/A	124.50 MHz (period = 8.032 ns)	dfb_theta[2]	temp_marks_rdaddress2[4]	clk	clk	None	None	7.744 ns
5	N/A	124.52 MHz (period = 8.031 ns)	dfb_theta[2]	temp_marks_rdaddress2[5]	clk	clk	None	None	7.743 ns
6	N/A	124.52 MHz (period = 8.031 ns)	dfb_theta[2]	temp_marks_rdaddress2[6]	clk	clk	None	None	7.743 ns
7	N/A	124.53 MHz (period = 8.030 ns)	dfb_theta[2]	temp_marks_rdaddress2[7]	clk	clk	None	None	7.742 ns
8	N/A	124.53 MHz (period = 8.030 ns)	dfb_theta[2]	temp_marks_rdaddress2[8]	clk	clk	None	None	7.742 ns
9	N/A	126.34 MHz (period = 7.915 ns)	dfb_theta[3]	temp_marks_rdaddress2[1]	clk	clk	None	None	7.627 ns
10	N/A	126.34 MHz (period = 7.915 ns)	dfb_theta[3]	temp_marks_rdaddress2[2]	clk	clk	None	None	7.627 ns

图 3-40 原始的少于 17 个特征点的打分模块前 10 条最大延时路径

Flow Status	Successful - Thu Feb 04 20:53:05 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	filter9_marker_total_grade
Top-level Entity Name	filter9_marker_total_grade
Family	Stratix II
Device	EP2S90F1020C5
Timing Models	Final
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	485 / 72,768 (< 1 %)
Dedicated logic registers	394 / 72,768 (< 1 %)
Total registers	394
Total pins	145 / 759 (19 %)
Total virtual pins	0
Total block memory bits	527,020 / 4,520,448 (12 %)
DSP block 9-bit elements	2 / 384 (< 1 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

图 3-41 新的少于 17 个特征点的打分模块资源占用率

	Slack	Actual fmax (period)	From	To	From Clock	To Clock	Required Setup Relationship	Required Longest P2P Time	Actual Longest P2P Time
1	N/A	114.64 MHz (period = 8.723 ns)	dfb_theta[2]	temp_marks_rdaddress2[1]	clk	clk	None	None	8.445 ns
2	N/A	114.64 MHz (period = 8.723 ns)	dfb_theta[2]	temp_marks_rdaddress2[2]	clk	clk	None	None	8.445 ns
3	N/A	114.65 MHz (period = 8.722 ns)	dfb_theta[2]	temp_marks_rdaddress2[3]	clk	clk	None	None	8.444 ns
4	N/A	114.65 MHz (period = 8.722 ns)	dfb_theta[2]	temp_marks_rdaddress2[4]	clk	clk	None	None	8.444 ns
5	N/A	114.93 MHz (period = 8.701 ns)	dfb_theta[2]	temp_marks_rdaddress2[5]	clk	clk	None	None	8.423 ns
6	N/A	114.93 MHz (period = 8.701 ns)	dfb_theta[2]	temp_marks_rdaddress2[6]	clk	clk	None	None	8.423 ns
7	N/A	114.93 MHz (period = 8.701 ns)	dfb_theta[2]	temp_marks_rdaddress2[7]	clk	clk	None	None	8.423 ns
8	N/A	114.93 MHz (period = 8.701 ns)	dfb_theta[2]	temp_marks_rdaddress2[8]	clk	clk	None	None	8.423 ns
9	N/A	116.69 MHz (period = 8.570 ns)	dfb_theta[2]	temp_marks_rdaddress2[0]	clk	clk	None	None	8.288 ns
10	N/A	116.69 MHz (period = 8.570 ns)	dfb_theta[2]	temp_marks_rdaddress2[9]	clk	clk	None	None	8.288 ns

图 3-42 新的少于 17 个特征点的打分模块前 10 条最大延时路径

比较图 3-39 和图 3-41，可以发现新的打分模块占用的 ALUTs 比原设计略有增加，主要原因是为了兼顾 m_realMNum=1 的特殊情况，逻辑变得复杂；registers 有所减少，主要原因是去掉了一些寄存单元。

比较图 3-40 和图 3-42，新的打分模块的最高时钟频率比原设计降低了近 10MHz，主要也是因为新的打分模块逻辑更加复杂。值得注意的是，图 3-40 的前 8 条最大延时路径与图 3-42 的前 8 条最大延时路径是一一对应的，而新的打分模块没有修改这 8 条路径。这也验证了 QuartusII 的综合及布局布线的一定程度上的随机性，但只要约束在设计允许的范围内就可以接受。

3.12 现场指纹特征点多于 16 个的打分模块 fitler_marker_up 的设计

filter_marker_up 位于图 3-3 的打分筛选模块内。filter_marker 选通 fitler_marker_up 后，offset_read_valid 也就连到了 fitler_marker_up 的 offset_read_valid_up。原设计的 filter_marker_up 没有包含 offset_read_valid_up 的

生成。由于后面的 fifo_write 要求对应于每一个有效基偏移量的 offset_read_valid 至少领先于 match 一个时钟周期有效，而且 filter_marker_up 是流水线结构的，延迟是确定的，所以把前面传来的通知打分信号 mark_up_valid 寄存一个时钟后，输出到 offset_read_valid_up，这样 offset_read_valid_up 提前 match_up（现场指纹特征点多于 16 个的打分模块的匹配信号）3 个时钟有效。如图 3-43 所示，这样设计使得对下一个有效基偏移量的处理不再依赖于 match，提高了数据处理速度。

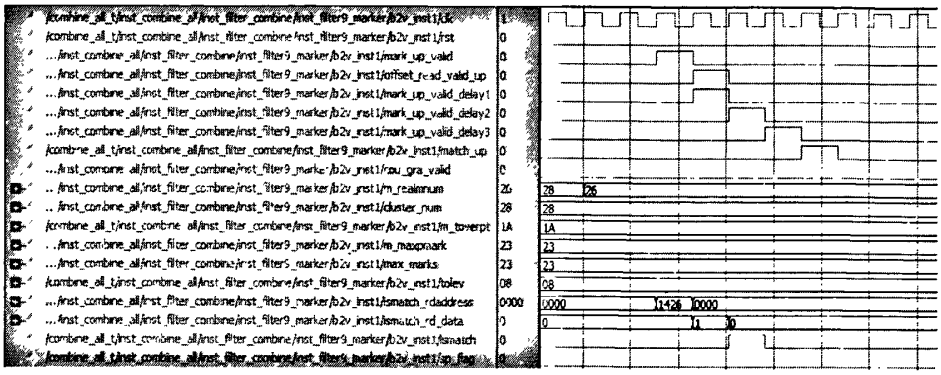

图 3-43 对现场指纹特征点多于 16 个的打分模块的功能仿真

第四章 片外 SRAM 控制器的设计

4.1 片外 SRAM 概述

SRAM 的全称是 Static Random Access Memory。它的特点是具有静态存取功能，不需要刷新电路即能保存它内部存储的数据。因此 SRAM 具有可记忆性和性能高的特点，广泛应用于电子领域。很多厂家也相继推出了高性能的 SRAM 芯片。

我们课题使用 IDT 公司的 IDT71V416S10PHZ0029P，它是异步 SRAM，有以下主要特点：

·256K×16 的先进高速 CMOS 静态 RAM。

·一个片选信号和一个输出使能管脚。

·双向的数据输入输出直接与 LVTTL 兼容。

·无效片选信号降低功耗。

·低字节和高字节使能引脚[7]。

该 SRAM 的管脚分布图如图 4-1 所示

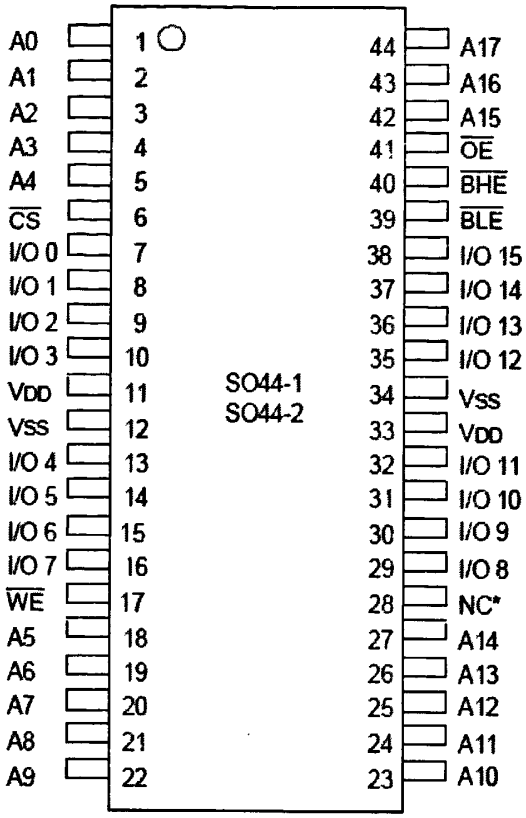

图 4-1 IDT71V416S10PHZ0029P 的管脚分布[7]。

BHE#和 BLE#分别是高字节和低字节的使能信号，低电平有效，本设计中不涉及字节粒度的操作，因此在设计 PCB 板时将它们接地。

4.2 片外 SRAM 控制器 sram 的整体规划

sram 分为 4 个子模块：SRAM 写控制、SRAM 读控制、SRAM 核心控制和与 SRAM 的接口。如图 4-2 所示：

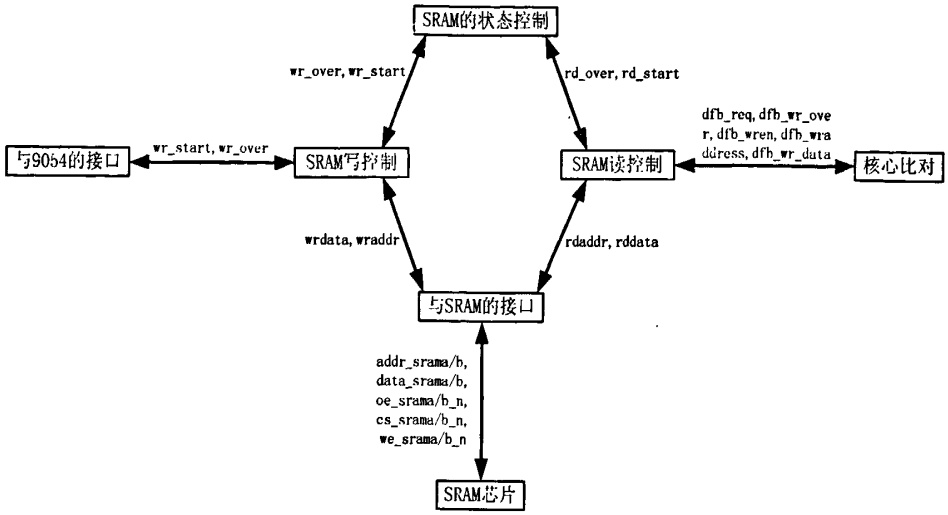

图 4-2 片外 SRAM 控制器 sram 的框图

本课题需要较大容量的库指纹缓冲，FPGA 里的 RAM 资源不够，因此在 PCB 板上焊了四块 SRAM，每块 SRAM 的数据线有 16 位。两块 SRAM 为一组，每组暂存 2000 枚库指纹。它们的地址线和控制线连在一起，分别存放库指纹特征点的高/低 16 位数据。系统启动时 SRAM 是空的，FPGA 把 PLX9054 发来的库指纹数据转存至 SRAM，等到核心比对模块需要读取库指纹时，再从 SRAM 里读出，转交给核心比对模块。

为提高传输效率，这两组片外 SRAM 采用乒乓操作。在写 A 组 SRAM 时，SRAM 控制器从 B 组 SRAM 中读待比对的库指纹。等到 A 组 SRAM 被写满，B 组 SRAM 被读空，SRAM 控制器再从 A 组 SRAM 中读库指纹数据，向 B 组 SRAM 中写入库指纹数据。这样一来，核心比对模块不间断读入库指纹特征点，比对也可以不间断进行。

4.3 SRAM 写控制模块 sram_write

本模块采用 SRAM 芯片手册中的第一种方式写，如图 4-3 所示：

Timing Waveform of Write Cycle No. 1 ($\overline{WE}$ Controlled Timing)[1,2,4]

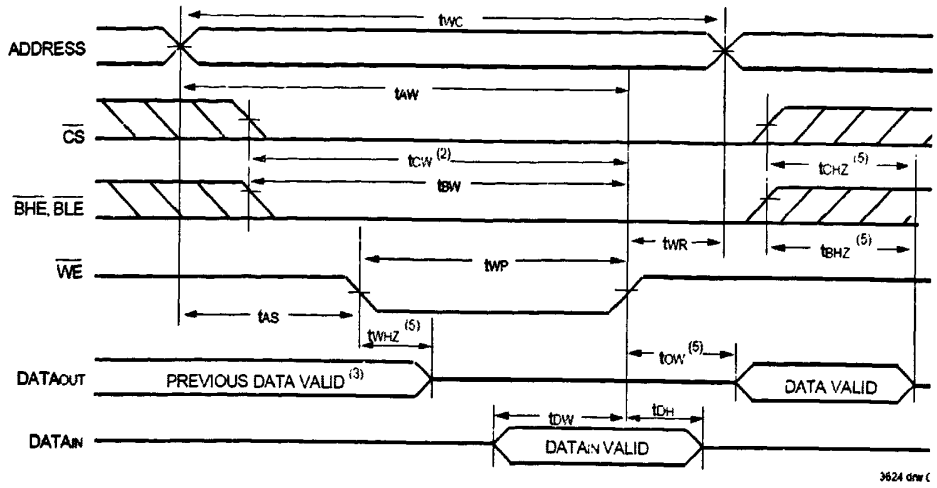

图 4-3 SRAM 的写时序图[7]

图 4-3 中，t_{wc} 是数据写周期，它的最小值决定了写数据的最大速率，对 PCB 板上的 SRAM 而言，它的最小值是 10ns。t_{AW} 是写操作结束前地址有效时间，最小值是 8ns。t_{cw} 是片选信号 CS#有效到写操作结束的时间，最小值是 8ns。t_{bw} 与 t_{cw} 类似，是高低字节 BHE#、BLE#选定信号有效到写操作结束的时间，最小值也是 8ns。t_{AS} 是地址建立时间，最小值是 0，也就是说，只要保证地址信号在 WE#的下降沿之前有效即可。t_{wp} 是写信号脉冲宽度，最小值是 8ns。t_{WHZ} 的意义是从 WE#的下降沿到数据总线变成高阻的时间，因为之前 SRAM 可能工作在读状态，OE#有可能是低电平，它的最大值是 6ns。t_{DW} 是 WE#的上升沿之前，数据输入保持稳定的时间，也即数据的建立时间，最小值是 5ns。t_{DH} 是数据的保持时间，最小值是 0。有一点需要注意的是，如果在写操作之前，SRAM 工作在读状态，那么 t_{WP} 必须大于 $t_{WHZ}+t_{DW}$ 以保证数据总线由输出态变为稳定的输入态，如果 SRAM 持续工作在写状态，那么 t_{WP} 则不受此限制。t_{OW} 是当 OE#=0 时，从 WE#的上升沿到输出端数据稳定所需时间，最小值是 3ns。t_{WR} 是地址的保持时间，最小值是 0。t_{CHZ} 和 t_{BHZ} 分别是从 CS#和 BHE#/BLE#的上升沿到数据线变为高阻的时间，它们的最小值都是 5ns。

综上所述，要将数据准确写入 SRAM，需要满足 7 个条件：

1. 数据写周期 t_{wc} 不得小于 10ns。
2. 在 WE#下降沿之前给出有效地址信号。
3. WE#保持低电平的时间 t_{wp} 不得小于 8ns。
4. 数据的建立时间 t_{DW} 不得小于 5ns。
5. 有效地址信号保持到在 WE#的上升沿之后。
6. 有效数据信号保持到在 WE#的上升沿之后。
7. 数据在 WE#的上升沿写入。

因此，写 SRAM 的过程最少可简化成 4 步：

1. 地址线给出有效的地址。

2. WE#给出下降沿。

3. WE#给出上升沿，WE#的低电平时间不得小于 5ns。

4. 释放地址线和数据线，此时距给出有效地址时间不得小于 8ns。

因此，写数据最少需要 3 个时钟实现，本地总线工作在 33M，核心比对模块工作在 66M， 在 DMA 传输模式下，本地总线的极限数据传输率是 33M×4/5*4=105.6Mbps，如果 SRAM 控制单元用 66M 时钟实现，数据传输率最高是 66M/3*4=88Mbps，低于本地总线极限数据传输率，需要很大容量的 FIFO，这是本课题不允许的。因此需要提高 SRAM 控制单元的工作频率。选择本地总线 33M 时钟的 5 倍频，即 165M 时钟，作为 SRAM 控制器的工作时钟。每过 5 个 165M 时钟完成一个数据的传输， SRAM 的最高写速率能够达到 165Mbps，可以满足本课题要求。

SRAM 写控制模块的状态机如图 4-4 所示：

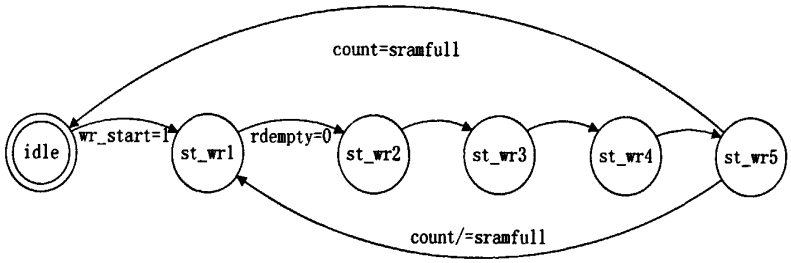

图 4-4 写 SRAM 的状态机

加载网表后，状态机停留在 idle，wr_data、wr_addr 都是全零的，wren_n 是'1'，都是无效的。等待 SRAM 控制单元核心传来的 wr_start，wr_start='1'表明启动装填一组 SRAM，状态机随即进入 st_wr1。

SRAM 写控制模块包含一个 FIFO，FIFO 用来做数据在不同时钟域间的交互，因为从 9054 来的数据工作在 33M 时钟，而控制 SRAM 的状态机是工作在 165M 时钟下的。状态机在 st_wr1 检测到 rdempty='0'得知当前 FIFO 里有数据，随后地址计数器 count 的值赋给 wraddr，然后 count+1,启动写一个数据的操作，进入 st_wr2。

从 st_wr2 无条件进入 st_wr3，同时将从 FIFO 里读出的数据赋给 wr_data，wren_n 由'1'变到'0'。

在 st_wr3 和 st_wr4，wr_addr、wr_data 和 wren_n 保持不变。

在 st_wr5，wren_n 产生上升沿，检查 count 是否等于 sramfull，sramfull 是预定义的 SRAM 容量，调试时为了方便可以设得小一些。count=sramfull 表明 SRAM

满了，回到 idle，否则回到 st_wr1，这样 wr_addr、wr_data 的变化必定发生在 wren_n 上升沿之后。

如图 4-3 所示，系统复位后进入 write_a1。如果 rdempty='0'，说明 9054 已把库指纹数据写入 FIFO 中，状态机转移到 write_a2，同时从 FIFO 读出数据并将 count 加 1，表示向 SRAM 里写入一个数据。

Quartus II 的嵌入式逻辑分析仪把要测试的信号存在 FPGA 内部的存储器中，利用 JTAG 口返回主机，可见嵌入式逻辑分析仪的引入，会在一定程度上影响所测工程的性能。所以，对于要观察的信号及其取样点数，还应遵循"够用就好"的原则。

图 4-5 是利用 QuartusII 9.1 的嵌入式逻辑分析仪得到的 sram_write 的波形。

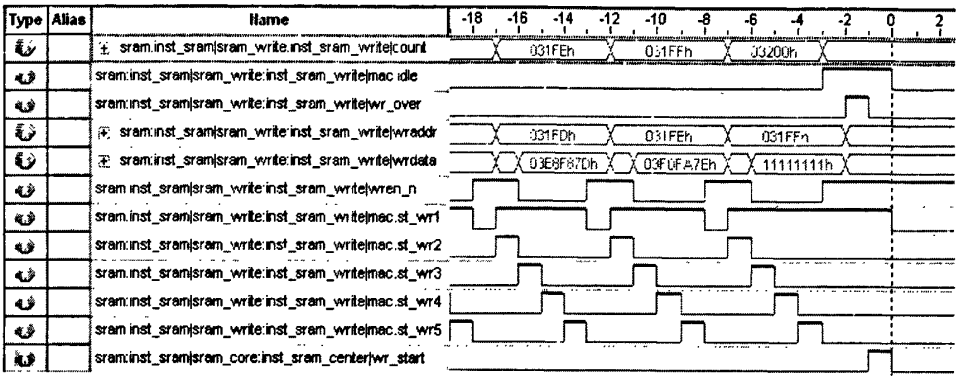

图 4-5 SRAM 写控制单元实测波形

4.4 SRAM 的读控制模块 sram_read

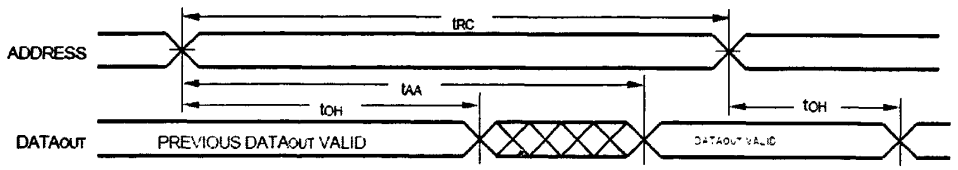

图 4-6 SRAM 读时序[7]

芯片手册里有两种读 SRAM 的方法，本课题采用第一种，如图 4-7 所示。

t_{RC} 是 SRAM 读周期，最小值是 10ns。t_{AA} 是从地址有效到数据稳定输出的间隔，最大值是 10ns。t_{OH} 是地址的变化反映到数据输出的时间，最小值是 4ns。

在读一个数时，保持 CS#、BLE#、BHE#和 WE#为'1'，OE#为'0'，给出有效的地址后，经过 t_{AA} 后 SRAM 给出有效数据。

SRAM 控制单元的工作时钟周期是 1/165≈6.1ns，因此在给出有效的地址后要等待至少两个时钟才能得到有效的数据，但实测等待时间要远远多于两个时钟，根据实测，等待 8 个时钟后，输出数据比较可靠。

SRAM 的模块读控制模块 sram_read 的状态机如图 4-7 所示：

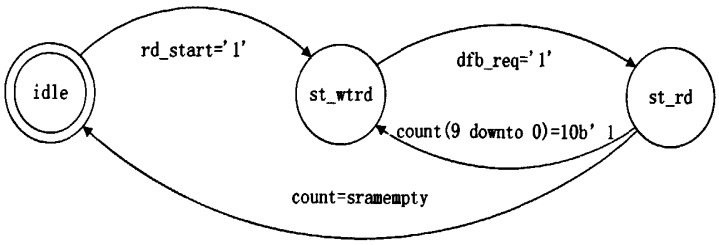

图 4-7 读状态机状态转移图

系统复位后状态机进入 idle，等到从 sram_core 传来的 rd_start='1'表示有一组 SRAM 已满，就转移到 st_wtrd，等到核心比对模块发读一枚库指纹请求（dfb_req='1'），进入 st_rd。

count 是读地址计数器，一枚库指纹占 128 个存储单元，开始读一枚库指纹时，count 的低 10 位是全 0，每过一个时钟 count 加 1，每过 8 个时钟，count 的第 4 位翻转一次，表示读出一个特征点，此时将 dfb_wren 置'1'，把从 sram_sram 返回的 rddata 赋给 dfb_wr_data，把将 count 寄存的第 10 至第 4 位后得到的 dfb_wraddress_reg 赋给 dfb_wraddress，写入边产生模块的乒乓 RAM。

dfb_wraddress_reg 是全'1'时表示读完一枚库指纹，此时置 dfb_wr_over 为'1'，如果这时当前这组 SRAM 空了（count=sramempty），那么回到 idle，同时发 rd_over 更新 sram_core 状态，反之回到 st_wtrd。

图 4-8 是 sram_read 的实测波形：

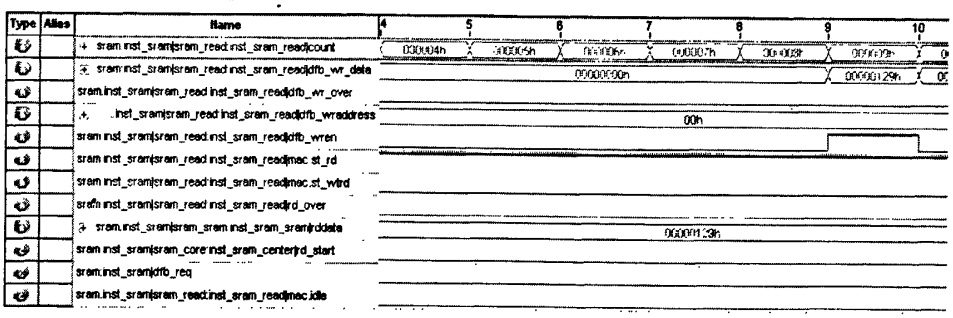

图 4-8 sram_read 的实测波形

4.5 SRAM 核心控制模块 sram_core

sram_core接收sram_write、sram_read传来的wr_over和rd_over，产生wr_start和rd_start，分别传给sram_write和sram_read，通知他们更新状态；并产生wr_a、wr_b、rd_a、rd_b传给与SRAM的接口sram_sram，作为A/B组的写/读选通信号。

SRAM核心控制模块的状态机如图4-9所示：

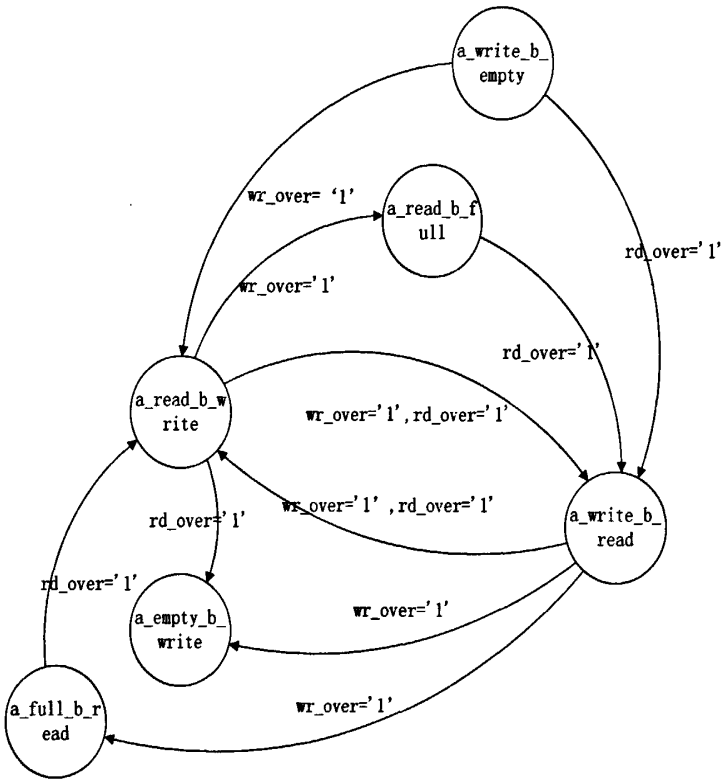

图4-9 sram_center的状态转移图

上电后，sram_core状态机进入a_write_b_empty，意义是写A组SRAM，B组SRAM是空的，sram_write写满A组SRAM后，发wr_over ='1'，状态机转移到a_read_b_write。

在a_read_b_write，SRAM控制器读A写B，如果在某一时刻，wr_over和rd_over都有效，就转移到a_write_b_read——写A读B；如果仅wr_over有效，就进入a_read_b_full——读A，同时B满；如果仅rd_over有效，就进入a_empty_b_write——写B，同时A空。

在a_empty_b_write，等到wr_over='1'，随后进入a_write_b_read。

在a_read_b_full，等到rd_over='1'，随后进入a_write_b_read。

在 a_write_b_read，如果在某一时刻，wr_over 和 rd_over 都有效，就转移到 a_read_b_write；如果仅 wr_over 有效，就进入 a_full_b_read；如果仅 rd_over 有效，就进入 a_write_b_empty。

在 a_full_b_read，SRAM 控制器读 B，同时 A 满，等到 rd_over='1'，随后进入 a_read_b_write。

在写或读某组 SRAM 时，sram_core 有效该组的读或写选通信号。以 A 组为例，如写或读 A 组时，置 wr_a 或 rd_a 为'1'。B 组同理。

如果在某一状态下，某组 SRAM 是空的，下一状态要写该组 SRAM，就在状态转移的同时置 wr_start 为'1'，例如在 a_empty_b_write 向 a_write_b_read 跳转的同时置 wr_start 为'1'。

如果在某一状态下，某组 SRAM 是满的，下一状态要读该组 SRAM，就在状态转移的同时置 rd_start 为'1'，例如在 a_full_b_read 向 a_read_b_write 跳转的同时置 wr_start 为'1'。

图 4-10 是 sram_core 的实测波形。

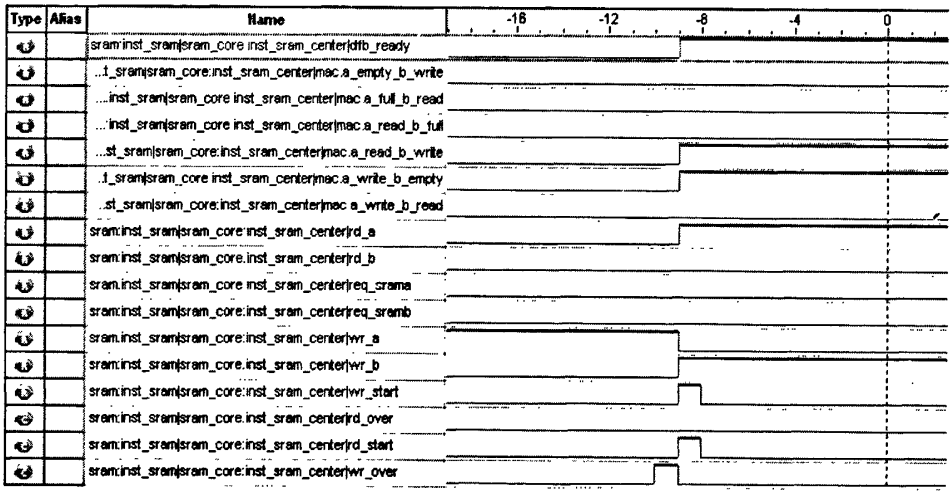

图 4-10 sram_core 的实测波形

图 4-10 中，dfb_ready 发给核心比对模块。如果当前时刻某一组空，同时在写另一组，就将 dfb_ready 置'0'，其它情况下 dfb_ready 是'1'。dfb_ready='1'表示 SRAM 可读。读空 A/B 组 SRAM 时，置 req_srama/req_sramb 为'1'，用于生成中断申请主机发一批库指纹。

4.6 与 SRAM 的接口 sram_sram

sram_sram 连接 sram_write、sram_read、sram_core 和 SRAM 芯片。sram_sram 接收 sram_write 和 sram_read 发来的地址、数据和控制信号，读写 SRAM，并把读出的数据交给核心比对模块。

sram_write 发给 sram_sram 的控制信号如下：

wraddr 和 wrdata：分别是写 SRAM 的地址和数据；wren_n：写使能信号，wren_n='0'时当前的地址和数据有效。

sram_read 发给 sram_sram 的控制信号如下：

rdaddr：读 SRAM 的地址。

sram_core 发给 sram_sram 的信号如下：

wr_a/wr_b：选通 A/B，进行写操作；rd_a/rd_b：选通 A/B，进行读操作。

sram_sram 发给 SRAM 的信号如下：

data_srama/data_sramb：连到 A/B 组 SRAM 的数据信号，既可做输入也可做输出；addr_srama/addr_sramb：连到 A/B 组的地址信号；cs_srama_n/cs_sramb_n：A/B 组的片选信号，即 CS#,低电平有效；we_srama_n/we_sramb_n：A/B 组的写使能，即 WE#；oe_srama_n/oe_sramb_n：A/B 组的输出使能信号，即 OE#，读 SRAM 时要拉低 OE#，OE#不影响 SRAM 的写操作，本设计中 oe_srama_n/oe_sramb_n 恒为'0'。

图 4-11 是 sram_sram 的实测波形。

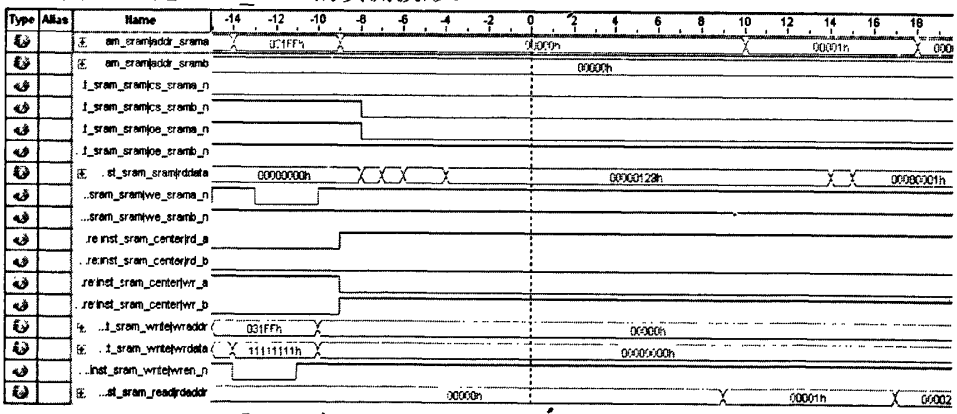

图 4-11 与 SRAM 的接口 sram_sram 的功能仿真

对于某一组 SRAM，某一时刻如果没有对它读或写，就把它的 CS#置为高电平，降低功耗和发热量。

第五章 本地总线与9054的接口

5.1 9054概述

PCI9054是32位、33MHz的通用PCI总线控制器专用芯片。该芯片符合PCI总线规范2.2版，突发传输速率能达到132MB/s。Local总线支持复用/非复用的32位地址/数据，可为M模式、C模式、J模式中的一种。PCI9054内部有六个可编程的FIFO，实现零等待突发传输及局部总线和PCI总线间的异步操作。PCI9054支持主模式(initiator)、从模式(target)、DMA传输方式，可用于适配卡和嵌入式系统。其工作电压为3.3V，频率33.3MHz，局部总线频率可达50MHz。PCI9054的主要特点如下：

1. 符合PCI V2.1、V2.2规范。兼容PCI V2.2电源管理规范；

2. 采用通用总线主控接口。包含两个独立的DMA通道；

3.支持PCI双地址周期(DAC)。地址空间可达4GB，支持TYPE0，TYPE1配置周期；

4. 内含可编程中断控制器。能实现可编程的突发传输操作的6个零时间等待可编程FIFO，8个32bit的Mailbox寄存器和2个32bit的Doorbell寄存器；

5. 支持局部总线与Motorola MPC860、PowerQUICC和Intel I960系列以及IBM的PPC401等CPU直接相连；

6. 支持局部总线与PCI总线异步工作，可编程控制局部总线等待状态，支持可编程预读取计数器[8]。

5.2 通信模式的选择

5.2.1 选择C模式的原因

9054可被配置于3种工作模式：M模式、C模式和J模式。

M模式是应用于Motorola MPC 850/860 Power QUICC 和POWerPC 80x/82x的32位工作模式，是大端的，地址线和数据线不复用。较多的应用于通信领域，但是一般的场合不使用。

C 模式是地址线和数据线不复用的 32 位工作模式，是小端的，开发难度较小，应用最广，资料也最多。

J 模式是地址线和数据线复用的 32 位工作模式。开发难度较大，资料也少。

综上所述，选择 C 模式应用于本课题。

在设计 PCB 板时，没有将 9054 的 Mailbox、Doorbell 等寄存器的控制端连到 FPGA 上，因此 FPGA 不适合作为主设备。同时为了防止本地总线上的冲突，本课题将 FPGA 只作为从设备，每次返回一批匹配库指纹 ID 和申请一批库指纹前，FPGA 把连到 9054 的中断线 LINT#拉低，应用程序接收到中断信号后，由应用程序采用 PCI TARGET 模式，引导 9054 读回匹配库指纹 ID 或向 FPGA 传入一批库指纹，9054 作为本地总线的主设备。

5.2.2 C 模式的工作流程

图 5-1 和图 5-2 分别是 PCI Target Write 和 PCI Target Read 的通信流程。

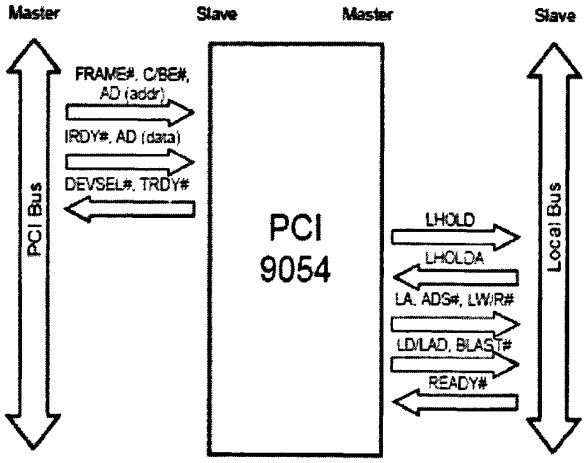

图 5-1 PCI Target Write 的通信流程 [9]

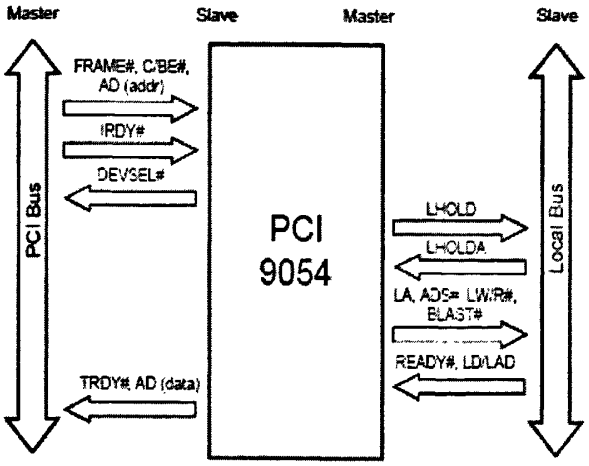

图 5-2 PCI Target Read 通信流程 [9]

在 PCI Target 模式下，在发起一次传输前，PCI9054 首先判断本地总线是否被占用，如果本地总线空闲（LHOLD='0'），那么就由 9054 驱动 LHOLD 至高电平，随后等待本地总线从设备返回 LHOLDA，LHOLDA 为'1'表明从设备已经做好准备接收数据。9054 收到 LHOLDA 后,将 ADS#拉低，开启一次传输。具体的时序如图 5-3 和图 5-4 所示。

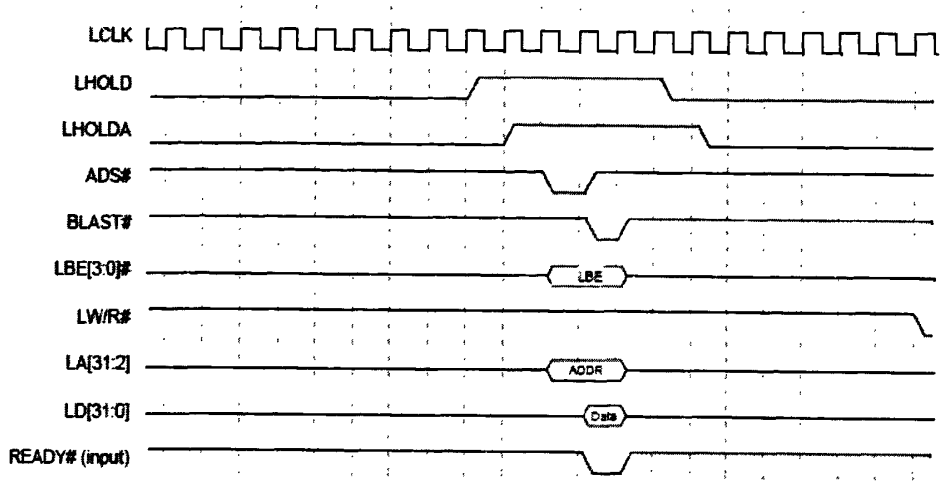

图 5-3 PCI Target Write 单周期写（32 位本地总线）[9]

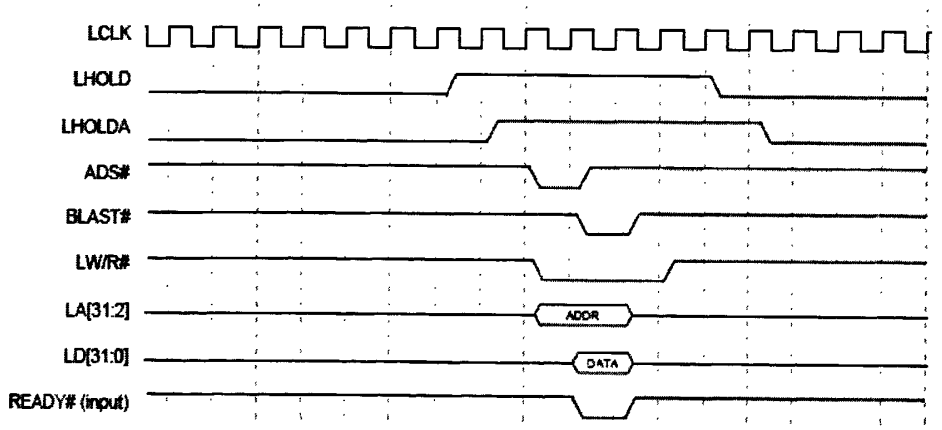

图 5-4 PCI Target Read 单周期读（32 位本地总线）[9]

图 5-3 中,在 9054 检测到 LHOLDA 为'1'时,将 ADS#置为'0',同时将 LW/R#置'1'，表明本传输周期是一个写周期，同时在 LA[31:2]上输出地址，并将 LBE[3:0]#置为有效值，LBE[3:0]#是字节使能信号，低电平有效，本地总线工作在 32 位小端模式时，LBE[3:0]对应 LD[31:0]的 4 个字节，LBE[0]#对应 LD[7:0]。一个时钟后在 LD[31:0]上输出要写入从设备的数据。因为是单周期写，第一个数据也是最后一个数据,所以置 BLAST#为'0',表示本次地址数据期结束。READY#由从设备发出，写周期中 READY#为'0'表示从设备已经将数据写入内部。主设备收到有效的 READY#后完成本次写周期。

图 5-4 中的单周期读与图 5-3 类似，所不同的是在拉低 ADS#的同时拉低 LW/R#，以表示本传输周期是一个读周期，LD[31:0]这时由从设备驱动至 LA[31:2]和 LBE[3:0]#共同指定的地址上的值，图 5-4 中的 READY#与图 5-3 中的含义有所不同，图 5-4 中的 READY#='0'表示当前的 LD[31:0]有效。

5.3 对 PCB 板布线的改动

原始 PCB 板中，FPGA 与 9054 虽然都工作在 33M 时钟，但是它们的时钟却是取自两块不同的晶振，实际上是异步工作的。这样给系统带来以下两点不利：

（1）异步时钟域间的同步器占用 FPGA 的资源，而且也增加了设计的困难。

（2） 由于存在异步时钟域间的同步，本地总线上数据传输效率降低，影响了加速卡系统的工作速度。

综上所述，尝试通过改变 PCB 板的时钟连线使 FPGA 和 9054 工作于同一个时钟域。由于本课题的 FPGA 是 PBGA 封装的，引脚以阵列状密布于芯片下方，飞线的难度较大，因此把 9054 与原来的晶振断开，从驱动 FPGA 的晶振引一条飞线连到 9054 的时钟输入端。这样做，有如下 3 点隐患：

（1） 驱动 9054 的时钟走的是飞线，介质是导线，而驱动 FPGA 的时钟走的是 PCB 板的时钟线，它们的长度与介电常数是不同的，造成延迟也会有区别。如果延迟很大，就会导致从 9054/FPGA 出来的信号不满足 FPGA/9054 的建立/保持时间，实质上依然是异步的。

（2） 晶振的驱动能力有限，如果没有同时驱动 9054 和 FPGA 的能力，会造成时钟品质的下降。

（3） 改动 PCB 布线后，可能会引入互相之间的干扰。

飞线后，不加同步器，9054 和 FPGA 之间可以正常工作，这说明它们之间可以看成是同步运行的，但飞线只是调试过程中的权宜之计，要想使系统真正稳定可靠的工作，需要修改 PCB 原理图重新制板。

5.4 本地总线上与 9054 的接口 local_9054 的设计

本课题的比对中需要的库指纹数据、现场指纹数据、查找表及各参量都要经由 local_9054 传入核心比对模块，匹配的库指纹 ID 也要经由 local_9054 返回主机。所以 local_9054 的设计既要包括读也要包括写。

图 5-5 是 local_9054 的状态转移图。

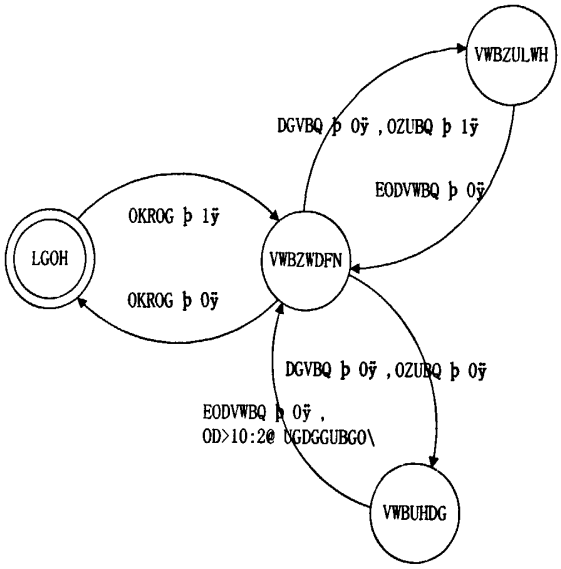

图 5-5 local_9054 的状态转移

系统加载后，状态机停在 idle，置 lholda='1'，等到 9054 发有效的 lhold（LHOLD='1'），进入 st_wtack，同时置 lholda='0'，响应 9054 对本地总线的申请。

在 st_wtack，等待地址数据开始信号 ads_n(ADS#)有效，如果在 ads_n='0'的同时 lwr_n（LWR#）为'0'表明本次地址数据期是一次读周期，转移到 st_read，如果 lwr_n='1'就转移到 st_write，如果没等到 ads_n='0'lhold 就变为'0'表示当前传输被取消，状态机返回到 idle。

在 st_write，如果 blast_n（BLAST#）是'0'，那么返回 st_wtack。

在 st_read，如果 blast_n 是'0'，并且 la[10:2]=rdaddr_dly，那么返回 st_wtack。

本课题涉及的都是 32 位数，所以忽略 LBE[3:0]#，为了方便，输出匹配库指纹 ID 时不设置 DP[3:0]，接收从 9054 发过来的数据时也忽略 DP[3:0]。

本课题将库指纹数据、现场指纹数据、查找表和相关参量等数据分为 8 类，用一个 3 位二进制数 hi_flag 标识，后面的参量分发模块 table_para 根据 hi_flag 得知输入的数据的类型，hi_flag 与配置数据及库指纹数据的对应关系如表 5-1 所示。

表 5-1: hi_flag 与数据类型对应关系

000	001	010	011	100	101	110	111
平方根反正切表数据	全局参量、阈值	th_grade 表数据	现场指纹三角形信息	现场指纹细节点数据	聚类分类信息	现场指纹边信息	库指纹特征点

为方便起见，hi_flag 取自 la[19:17]，la[16:2]所能表示的寻址范围是 32768 个 32 位数，由于写 SRAM 的地址由 SRAM 控制单元内部生成，所以本课题约

定 la[19:2]表示的每一枚库指纹的特征点的地址范围都是 0x38000 至 0x3807f，hi_flag 为“000”~“110”时，由 la[16:2]生成的 hi_count 的地址空间的范围也足以容纳每一种数据，不存在地址不够用的问题。

要返回的匹配库指纹 ID 存放于一个 FIFO 中，所以读数据时不需要用到地址信号 la，读匹配库指纹 ID 的具体做法是在 9054 要求读第一个数时，从 FIFO 中取数，以后每当有效的地址信号 la[10:2]发生变化时，说明 9054 要读下一个数，就从 FIFO 中再读出一个数。

rdaddr_dly 记录的是上一个有效的地址，具体的说，在收到第一个有效地址时，将它赋给 rdaddr_dly，以后每检测到有效地址变化，就将新的地址赋给 rdaddr_dly。这样每当有效的地址与 rdaddr_dly 不同时，就说明需要读取下一个匹配库指纹 ID。

当 blast_n（BLAST#）出现下降沿时，正好地址线上刚刚出现新的有效地址，而这时还未返回这个地址指向的数据，等到这个地址被写入 rdaddr_dly 时，正好从 FIFO 中读出的数据也出现在数据线上，所以在检测到 blast_n=‘0’，并且 la[10:2]=rdaddr_dly 时，退出 st_read，回到 st_wtack。

由 FPGA 返回给应用程序的中断有 3 个来源：A 组 SARM 读空、B 组 SRAM 读空和存放匹配库指纹的 FIFO 已满（实际上 FIFO 内还有存储空间，但为不打断流水而提前通知主机）。本地总线的中断线只有一条（LINT#），由于本 PCB 板的设计，FPGA 不能访问 9054 内部寄存器，也就不能通过设置不同的 Messagebox 或 Doorbell 等寄存器区分中断信息。所以在 FPGA 内部模拟了 Messagebox 寄存器：将 3 个中断源 lint_fifo、lint_srama 和 lint_sramb 相或后取反赋给 lint_n（LINT#），3 个中断源中任一个为‘1’时，lint_n=‘0’。

设计应用程序，当检测到本地总线上的中断后，首先读 FPGA 内部的模拟 Messagebox 寄存器。为了与匹配库指纹 ID 区分，将模拟 Messagebox 寄存器的地址设为本地总线的 Local Address Space 1 的首地址，即偏移量为 0，偏移量为 1 的是第一个匹配库指纹 ID，以此类推，设 FIFO 的容量是 PAF，那么偏移量为 PAF 的是最后一枚匹配库指纹的 ID。当 ads_n=‘0’，且 lwr_n=‘0’时，如果地址是 0，就把 lint_fifo、lint_srama 和 lint_sramb 选通到本地总线数据线的最低 3 位，应用程序可按位分析读出的模拟 Messagebox 寄存器得知中断来源；如果地址不是 0，那么按正常步骤从 FIFO 中读出匹配库指纹 ID 交给 9054，并由 9054 转发给主机。

如果读的是模拟 Messagebox 寄存器，那么发送 lack_fifo、lack_srama 和 lack_sramb，分别作为 lint_fifo、lint_srama 和 lint_sramb 的确认信号，各中断源

在收到有效确认信号后取消中断，从读模拟 Messagebox 寄存器到中断源取消中断的时间远小于应用程序处理中断的时间，所以不会发生误操作。

在读出最后一枚匹配库指纹的 ID 时，置 lrd_over 为'1'，通知 FIFO 控制器更新状态。

图 5-5 和 5-6 是本地总线上写和读的实测波形。

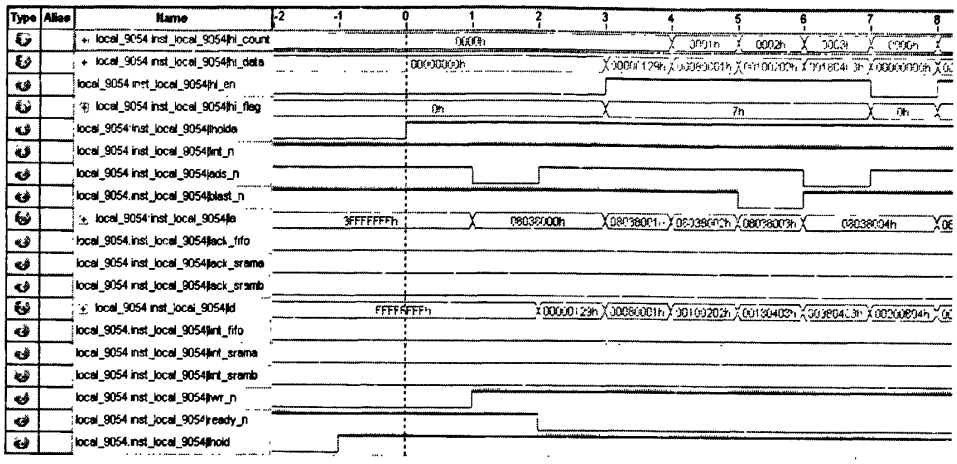

图 5-5 本地总线上写实测波形

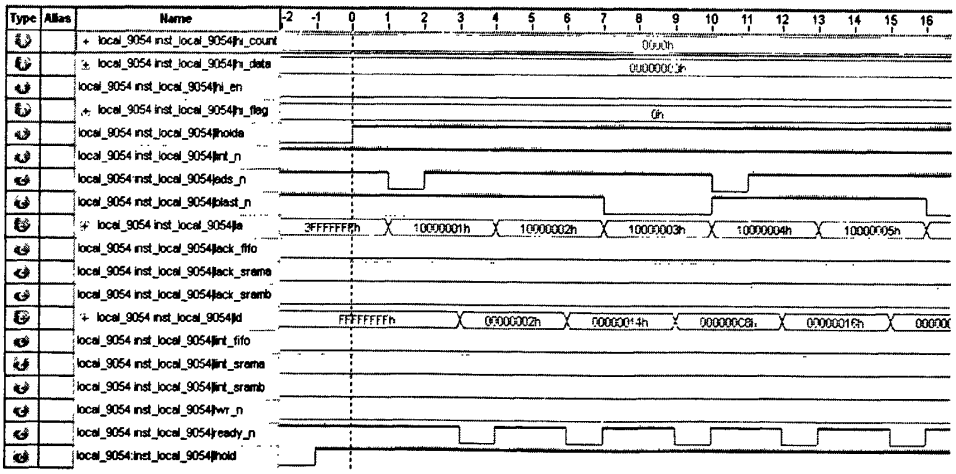

图 5-6 本地总线上读实测波形

第六章 验证和测试结果

6.1 FPGA 代码编译报告

在 QuartusII 9.1 环境下，优化选项选择“Balanced”,不选“Power-Up Don't Care”,闲置管脚设置成“As input tri-stated with bus-hold circuitry”，禁用嵌入式逻辑分析仪，得到的编译报告的资源占用情况如下：

Flow Status	Successful - Tue Jun 01 13:15:02 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Full Version
Revision Name	combine_all
Top-level Entity Name	combine_all
Family	Stratix II
Device	EP2S90F1020C5
Timing Models	Final
Met timing requirements	Yes
Logic utilization	71 %
Combinational ALUTs	33,034 / 72,768 (45 %)
Dedicated logic registers	33,894 / 72,768 (47 %)
Total registers	33894
Total pins	187 / 759 (25 %)
Total virtual pins	1
Total block memory bits	2,076,054 / 4,520,448 (46 %)
DSP block 9-bit elements	127 / 384 (33 %)
Total PLLs	2 / 12 (17 %)
Total DLLs	0 / 2 (0 %)

图 6-1 FPGA 上全部模块的联合编译报告

从图 6-1 可见，各项资源的占用率都比较合理，其中，总的逻辑资源占用率为 71%，说明资源得到了充分利用，且有足够的空余资源供编译器进行速度优化。

图 6-2 是利用 TimeQuest Timing Analyzer 得到的各时钟域正常工作的最高频率。

	Fmax	Restricted Fmax	Clock Name	Note
1	75.9 MHz	75.9 MHz	lclk	
2	77.92 MHz	77.92 MHz	clk_66M_Gen:inst_clk_66M_Gen|pllu:pllu_inst|altpll:altpll_component|_clk0	
3	193.72 MHz	193.72 MHz	sram:inst_sram|pll_33M_165M:inst_pll_33M_165M|altpll:altpll_component|_clk0	

图 6-2 用 TimeQuest Timing Analyzer 得到的各时钟域正常工作的最高频率

本课题共有 3 个时钟域：分别是本地总线上的 33M 时钟、核心比对的 66M 时钟和 SRAM 控制器的 165M 时钟，图 6-2 中从上到下分别是这 3 个时钟域的可以正常工作的最高频率，通常为使系统工作可靠，要求最高频率要实际工作频率高 5%，图 6-2 中的最高频率都满足这一要求。

6.2 与应用程序、驱动程序联合测试

6.2.1 测试程序的设计

为了方便查错，在 SRAM 控制器中，将每一组 SRAM 的容量设置为 100×128×32bit，也就是 100 枚库指纹。设置存放比对结果的 FIFO 控制器，每写入 FIFO100 个 ID 号引发一次本地总线中断。

目前还未完成传送库指纹的应用程序与指纹数据库的连接，所以还不能用实际的库指纹和现场指纹进行测试。为了简化测试，自定义一枚现场指纹，并把该现场指纹的不同序号的 100 个副本作为库指纹，如果设计正确，那么这 100 枚库指纹都能通过比对。

编写应用程序函数 WriteLibFps，完成传送 100 枚库指纹的工作。完成现场指纹、查找表及相关参量配置后，应用程序首先调用 WriteLibFps 两次，也就是把两组 SRAM 写满。然后应用程序从预先编辑好的文本文件中读出期望的加速卡返回的结果，等待加速卡产生中断。

加速卡产生中断后，应用程序首先判断该中断是否由本地总线上的 LINT# 引起。如果不是由 LINT#引起就忽略该中断；反之读 FPGA 内部的模拟 Messagebox 寄存器，得到中断原因（见 5.4 节）。

应用程序首先检测中断源 lint_fifo，如果 lint_fifo='1',说明加速卡要求返回匹配库指纹 ID，那么就从加速卡读回 ID 并与期望结果比较，如果发现某一 ID 与期望结果不同就报错，并终止比对。如果顺利完成对 lint_fifo 的处理，就再检测中断源 lint_srama 和 lint_sramb，如果 lint_srama 为'1'，调用 WriteLibFps，向加速卡写入 100 枚库指纹，如果 lint_srama 为'1'，就再次调用 WriteLibFps，向加速卡再写入 100 枚库指纹。

在应用程序中设置一个计数器记录已比对库指纹的数目，比对 1 亿枚后认为加速卡通过测试，完成比对。

图 6-3 是应用程序的工作界面。

图 6-3 应用程序工作界面

图 6-3 中，点击标题为“浏览”的按钮可以选择现场指纹、库指纹、查找表或其他参量的配置文件，选定的配置文件显示在旁边的文本框里。

开始比对前，首先点击“开始配置”用选定的现场指纹、查找表和其他参量的配置文件配置 FPGA 的寄存器和内部 RAM；然后点击“开始比对”，启动 1 亿枚库指纹的比对。“板卡状态”下方的文本框显示已比对多少枚指纹，最下方的进度条显示比对进度，进度条由空到满表示完成 10 万枚库指纹的比对，然后再次循环，这样做是因为，如果设置为比对 1 亿枚指纹后进度条充满，那么进度条要很长时间才动一点，不够生动。

“查看结果”和“复位”按钮的功能尚未完成。

6.2.2 测试结果

经测试，可以完成 1 亿枚库指纹的比对。时间大约花费 14 个小时，比对速度较慢，原因有以下 3 点：

（1） 测试时采用 PCI Target Single Write/Read 模式，这种方式要 CPU 参与完成，因此占用了 CPU 的处理能力，而且这种方法对本地总线的利用率很低。

（2） 为便于查错，片外 SRAM 乒乓缓冲和存放匹配库指纹 ID 的 FIFO 的容量都只有 100，造成比对进程频繁中断，核心比对模块的状态机在等待应用程序处理中断时，耗费了大量时间。

(3) 为便于查错，库指纹和现场指纹的特征点相同，匹配率是 100%，所有的库指纹 ID 都要返回主机，也增加了中断的频率。

对此的解决方案如下：

修改 9054 传输模式为 DMA。一方面， 9054 分担了 CPU 的工作；另一方面，提高了本地总线利用率。

修改片外 SRAM 乒乓缓冲和存放匹配库指纹 ID 的 FIFO 的容量，如每一组 SRAM 的容量最大可设置为 2000，存放匹配库指纹 ID 的 FIFO 由 FPGA 的内部 RAM 例化，因此更加灵活，可以根据库指纹容量和匹配率选择较合适的 FIFO 容量。

在实际的工作中，库指纹与现场指纹的匹配率较低，因此 FIFO 满（实际内部还有存储资源作为缓冲，见 5-4 节）引发中断的频率也较低。

第七章 总结与展望

7.1 总结

基于 FPGA 的指纹粗比对加速卡是自动指纹识别系统中的一个重要组成部分，它所完成的主要功能是粗比对和粗筛选，剔除大部分不匹配的指纹，将少数可能匹配的指纹交给主机程序，由主机程序完成精匹配工作。

本文完成了粗比对算法部分模块优化和改进，完成了 FPGA 与外围芯片的互通，进行了 RTL 级和验证并通过实测；硬件实现的性能、硬件仿真结果以及软件验证的结果表明硬件实现的性能可以满足设计要求。

7.2 展望

测试中采用了 9054 的 C 模式的单周期写/读，这是因为如果用 DMA 方式传输测试过程中主机会重启。距离实用化，还要完成下面的 3 个工作。

（1） 调试驱动程序并修正驱动程序的错误，实现用 DMA 传输数据。

（2） 改进应用程序，增加运行中复位和返回比对结果的功能。

（3） 添加与库指纹数据库的链接程序。

参考文献

[1] 李文亮，基于 FPGA 的指纹比对加速卡接口模块的设计与实现，北京邮电大学硕士学位论文，北京邮电大学图书馆论文档案室，北京邮电大学，2007 年

[2] 陈增茂，基于 FPGA 的指纹比对加速卡的设计与实现，北京邮电大学硕士学位论文，北京邮电大学图书馆论文档案室，北京邮电大学，2006 年

[3] 潘松，黄继业，EDA 技术与 VHDL 第 2 版，清华大学出版社，2007 年

[4] EDA 先锋工作室，吴继华，王诚，Altera FPGA/CPLD 设计（高级篇），人民邮电出版社，2005 年

[5] 李大伟，基于 FPGA 的指纹粗比对算法的优化与实现，北京邮电大学硕士学位论文，北京邮电大学图书馆论文档案室，北京邮电大学，2008 年

[6] ALTERA Corporation Stratix II Device Handbook, Volume 1, May, 2005 Ver.1.4

[7] IDT IDT71V416S/L Datasheet, May 1999

[8] 刘春河，指纹识别设备驱动的开发和车牌字符识别的 DSP 实现，北京邮电大学硕士学位论文，北京邮电大学图书馆论文档案室，北京邮电大学，2010 年

[9] PLX Technology PCI 9054 Data Book, Jan, 2000 Ver.2.1

致 谢

感谢蔡安妮教授，感谢蔡老师在我攻读硕士研究生期间给予的悉心指导、关爱与信任。虽然三年研究生生活短暂，但蔡老师严谨的学风、认真负责的学习和工作态度将使我受益终生！感谢庄伯金老师，庄老师不仅在我完成课题期间提了很多宝贵的意见，在读研的三年里，也以对科研一丝不苟的态度为我今后的工作和学习做出了榜样。在此向两位老师表示衷心的感谢！

感谢硬件教研室的刘春河同学，完成了驱动程序和应用程序的初步设计；感谢硬件教研室的徐艳园同学，完成加速卡 PCB 板的设计与制版；感谢硬件教研室的李大伟同学，完成核心比对算法的主要设计工作；感谢硬件教研室的张官兴、张凯、黄康莹等同学，他们在技术方面给与了我很多帮助，课题中很多问题的解决都来源于和他们的交流；感谢南晓明、胡翟、吴萌、彭旭、孟卿卿、季昊、梁焱睿等同学，他们陪我度过三年难忘的研究生生活！

另外还要感谢我的家人，感谢他们对我一如既往的关爱与支持！

最后对所有曾经支持、帮助过我的老师、同学、朋友致以诚挚的感谢！

攻读硕士学位期间发表的学术论文

[1] 刘恩茂，“用 FPGA 开发 SRAM 控制器”，中国科技论文在线，已经录用。